Memoir—Formal Specs and Correctness Proofs

John R. Douceur[†], Jacob R. Lorch[†], Bryan Parno[†], James Mickens[†], Jonathan M. McCune[‡]

[†]Microsoft Research, Redmond, WA
[‡]Carnegie Mellon University, Pittsburgh, PA

ABSTRACT

This tech report presents formal specifications for the Memoir system and proofs of the system's correctness. The proofs were constructed manually but have been programmatically machine-verified using the TLA+ Proof System.[3] Taken together, the specifications and proofs contain 61 top-level definitions, 182 LET-IN definitions, 74 named theorems, and 5816 discrete proof steps. The proofs address only the safety of the Memoir system, not the liveness of the system. Safety is proven by showing that a formal low-level specification of the Memoir-Basic system implements a formal high-level specification of desired behavior. The proofs then show that a formal specification of the Memoir-Opt system implements the Memoir-Basic system.

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Overview

This tech report presents formal specifications and safety proofs for the Memoir system. The specifications herin are written in the TLA+ language,[6] and the proofs are written in the TLA+ proof language.[7] Our hope is that a reader unfamiliar with TLA+ can easily understand the prose descriptions in this tech report, along with the textual comments embedded in the specifications and proofs. However, a thorough understanding of this tech report requires a solid knowledge of TLA+. We do not provide even a cursory tutorial of the language herein.

In TLA+, a specification is inductive. The spec describes a set of *state variables*, the initial values for these variables, and a set of *actions* that modify the variables. Each action is a relation between a pair of successive states. The temporally earlier state is called the *unprimed* state, and the temporally later state is called the *primed* state. (Within the language, the different states are identified by the absence or presence of a prime character following the state variable or expression.)

Our proofs are written in a hierarchical style advocated by Lamport.[4] Each progressive level of the proof contains sub-proofs, each of which proves a single proof step at the prior level. Our proofs contain a total of 5816 discrete proof steps, all in the service of proving 74 named theorems. Although these steps were all written manually, they have been programmatically machine-verified using the TLA+ proof system.[1–3]

1.1.1 Background—The Memoir System

Memoir is a generic framework for executing modules of code in a protected environment. In particular, Memoir guarantees not only privacy and integrity; it also guarantees *state continuity* across TCB interruptions. This means that, when a module pauses execution and returns control to the untrusted caller, and then later resumes execution, the module will resume from the same state it was in before it paused.

Understanding the formal specifications in this tech report requires a detailed understanding of the Memoir system, which this tech report does not provide. The reader is referred to our paper[8] describing the Memoir system. The level of knowledge contained in the cited paper will be assumed by the remainder of this tech report.

The associated paper introduces several terms of art with meanings that are specific to Memoir, such as "history", "history summary", and "authenticator". Herein, we supplement these terms with two others that represent intermediate values in the construction of an authenticator:

state hash: a secure hash of a public state and an encrypted private state

history state binding: a secure hash of a history summary and a state hash

Thus, an authenticator is a MAC of a history state binding.

1.1.2 Philosophy and Approach

One common way to formally address the correctness of a system is to state and prove particular properties that the system maintains. Closely related is the approach of proving that the system prevents a particular set of undesirable things from occurring. For instance, in Memoir, we might have asserted that the system is correct if it does not allow a rollback attack to set the service state to a previous state. Although this might seem intuitive, this particular property is is in fact too strong, because one could easily define a service that allows transitions to arbitrary previous states. Memoir does not prevent such a service from executing, and there no reason Memoir should be constrained to only run services that disallow repeated entries into the same state.

More importantly, there is a general problem with the approach of stating desirable and/or undesirable properties and proving that they do/don't follow. The problem is that there is no *a priori* reason to believe that any particular set of properties sufficiently captures the intended behavior of a system. Even if we were to modify the above non-rollback property to account for systems that allow state re-entry, there is no reason to believe that this is the only important property for the Memoir system to maintain. And, in fact, it is not the only important property: Memoir should also prevent a transition to any state that is not reachable by the service code. Even this is not sufficient, because the service may define multiple states that are independently reachable from the initial state, but that are mutually exclusive in any given execution sequence. We could continue adding and modifying properties, but there is no clear way to know when the set is sufficient to characterize the desired system behavior.

Instead of defining properties, we follow a proof approach encouraged by TLA+. This approach has four main parts, and often includes a fifth. The first part of the approach is to define a **high-level specification** that describes the intended semantics of the system. The high-level spec is small enough and simple enough that a knowledgeable reader should be able to examine the spec and easily determine whether its semantics are the right ones. It is, of course, possible to prove properties of the high-level spec, but the hope is that the high-level spec is straightforward enough that its desirability is readily assessable.

The second part of the approach is to define a **low-level specification** that describes the implementation of the system. Whereas a high-level spec typically describes abstract state at the semantic level, a low-level spec typically describes concrete state at the implementation level. Just as it should be easy to determine that the high-level spec describes desirable semantics, it should be easy to determine that the low-level spec accurately characterizes the real hardware and software that implements the system.

The third part of the approach is to define a **refinement** that describes how to interpret any given state of the low-level system as a corresponding state of the high-level system. This is a somewhat subtle concept, and we do not elaborate on it here, although the specific descriptions of refinements below (§ 1.2.3, § 1.3.4) may implicitly provide sufficient edification. A brief and surprisingly entertaining introduction to the topic of refinement is presented in the paper *Refinement in State-Based Formalisms*[5] by Lamport. For the very interested reader, the book *Specifying Systems*[6] describes the concept in depth.

The fourth part of the approach is where the rubber meets the road: a **proof of implementation** that shows that any behavior satisfying the low-level spec, when interpreted according to the refinement, also satisfies the high-level spec. Such a proof may (and ours does) show the mapping from particular actions in the low-level spec to particular actions in the high-level spec. This additional set of cross-spec correspondences provides further understanding of the relationship between the two specs, beyond merely that provided by the state-to-state correspondences established by the refinement.

The fifth (and the only optional) part the approach is to define and prove a set of **inductive invariants** that are maintained by the low-level spec. These inductive invariants may seem similar to the correctness properties we disparaged above; however, they are different in two important respects. First, and most importantly, they are completely in service to the proof of implementation. The only reason the inductive invariants are needed (if they even are) is as a necessary step in the process of proving that the low-level spec satisfies the high-level spec. Consequently, there is no danger that the set will be incomplete in some important way. If the invariants are sufficient to enable the proof of implementation, then the set is complete.

Second, it is not important for these invariants to be understood by a person who merely wants to be confident that the low-level system provides desirable semantics. Such a person need only understand the high-level spec, along with the abstract assertion that the low-level spec implements the high-level spec. This contrasts with the approach of defining desirable/undesirable properties, which of necessity must be understood by anyone wishing to know what the system is supposed to do. The invariants are important only to someone who wants to know *why* the low-level system satisfies the high-level semantics. It is often (but not always) the case that the essence of this why is in the definition of the inductive invariants.

For the particular case of Memoir, we define a single high-level spec but two low-level specs, one for Memoir-Basic and one for Memoir-Opt. We construct two refinements, one that maps from the Memoir-Basic spec to the high-level spec, and one that maps from the Memoir-Opt spec to the Memoir-Basic spec. We prove that the Memoir-Basic spec implements the high-level spec, which requires three inductive invariants. Proving these inductive invariants in turn requires stating and proving two more inductive invariants. We then prove that the Memoir-Opt spec implements the Memoir-Basic spec, which transitively implies that it implements the high-level spec. For this second implementation proof, no inductive invariants are necessary.

1.1.3 Assumptions

The proofs herein depend upon several assumptions. Some of these are realized as explicit assumptions using a TLA+ **ASSUME** statement. Others are realized implicitly in the definitions of various actions. The strongest assumptions we make are the following:

Assumption: Highly improbable events never occur.
Realization:

- The hash function is fully collision-resistant. See the explicit assumptions named *HashCollisionResistant* and *BaseHashValueUnique*.
- The MAC functions are fully collision-resistant and unforgeable. See the explicit assumptions named *MACCollisionResistant* and *MACUnforgeable*.
- Upon restarting, the arbitrary values in the computer's RAM will not contain an authenticator that is coincidentally equal to an authenticator that could be computed with the symmetric key

stored in the NVRAM. See the definitions of the *LL1Restart* and *LL2Restart* actions.

Assumption: The untrusted system cannot modify the contents of the NVRAM.

Realization:

- In the Memoir-Basic spec, the only action that changes the value in the NVRAM is *LL1PerformOperation*. See the definitions of *LL1Next* and all actions it disjoins.
- In the Memoir-Opt spec, the only action that changes the value in the NVRAM is *LL2PerformOperation*. See the definitions of *LL2Next* and all actions it disjoins.

Assumption: The SPCR can be modified only by resetting it or *extending* it. Extending means that the new value is a chained hash of the previous value in the SPCR with another value.

Realization:

- There are only three actions that modify the SPCR: *LL2PerformOperation*, *LL2Restart*, and *LL2CorruptSPCR*. See the definitions of *LL2Next* and all actions it disjoins.
- The *LL2PerformOperation* action extends the SPCR. See the definition of the *LL2PerformOperation* action and the *Successor* operator.
- The *LL2Restart* action resets the SPCR. See the definition of the *LL2Restart* action.
- The *LL2CorruptSPCR* action extends the SPCR. See the definition of the *LL2CorruptSPCR* action.

Assumption: The symmetric key stored in the NVRAM is unknown outside the trusted subsystem.

Realization:

- The only authenticators available to an attacker are (1) those previously returned by Memoir and (2) those the attacker can generate using a symmetric key other than the key stored in the NVRAM. See the definitions of *LL1CorruptRAM* and *LL2CorruptRAM*.

Assumption: The hash barrier stored in the NVRAM is unknown outside the trusted subsystem.

Realization:

- When an attacker extends the SPCR, the value for the extension cannot be constructed as a hash of any value with the hash barrier secret stored in the NVRAM. See the definitions of *LL2CorruptSPCR*.

In addition to the above strong assumptions, we use TLA+ **ASSUME** statements for several other purposes:

Type safety of primitives, parameters, and formalisms

- The primitive operators for hashing, message authentication codes, and symmetric cryptography are assumed to be type-safe. This is asserted by the explicit assumptions *BaseHashValueTypeSafe*, *GenerateMACTypeSafe*, *ValidateMACTypeSafe*, *HashTypeSafe*, *SymmetricEncryptionTypeSafe*, and *SymmetricDecryptionTypeSafe*.
- The service that the Memoir platform executes is assumed to be type-safe, as asserted by the explicit assumptions *ServiceTypeSafe* and *ConstantsTypeSafe*.
- Formalisms needed by the proof are assumed to be type-safe. See the explicit assumptions *HashCardinalityTypeSafe* and *CrazyHashValueTypeSafe*.

Correctness of primitives

- The MAC functions are assumed to be complete, meaning that every MAC generated with a key validates correctly with the same key, as asserted by the explicit assumption *MACComplete*.
- The MAC functions are assumed to be consistent, meaning that if a MAC validates correctly with a key, it must have been generated as a MAC with that same key, as asserted by the explicit assumption *MACConsistent*.
- The cryptographic functions are assumed to be correct, meaning that decryption is the inverse of encryption with the same key, as asserted by the explicit assumption *SymmetricCryptoCorrect*.

Formalisms

- One consequence of the strong collision-resistance of the hash function is that the result of any hash chain has a well-defined count of hashes that went into its production. We formalize this as the *cardinality* of the hash using the operator *HashCardinality* along with a set of explicit assumptions: *HashCardinalityAccumulative*, *BaseHashCardinalityZero*, and *InputCardinalityZero*.

- When a flag in the Memoir-Opt NVRAM indicates that the SPCR should contain the value *BaseHashValue* but it in fact contains some other value, we represent the logical value as a formalized *CrazyHashValue*. This value is assumed to be unequal to any other hash value by the explicit assumption *CrazyHashValueUnique*.
- The *HistorySummariesMatch* predicate is defined recursively, but the current version of the prover can neither handle recursive operators nor tractably support proofs using recursive function definitions. Therefore, we define the operator indirectly, by using the explicit assumption *HistorySummariesMatchDefinition*.

One final—and very important—assumption of this tech report is the correctness of our inductive reasoning. As of this writing, the current version of the TLA+ Proof System is unable to verify the proof step that ties together a base case and an induction step into an inductive proof. We thus depend upon human reasoning skills to ensure that this final step is correct for all proofs that use induction. This includes:

- the use of the *Inv1* rule in the proofs of type safety for our three specs and in the proofs of the inductive invariance of four invariants
- the use of the *StepSimulation* rule in the two implementation proofs
- the final step in the *HistorySummariesMatchUniqueLemma*, which uses non-temporal inductive reasoning

1.2 Memoir-Basic

Since Memoir is a platform that supports arbitrary services, our high-level spec declares the service to be an undefined function that maps a state and a request to a state and a response. More precisely, the spec partitions the service state into a public portion and a private portion, with the intent that only the private portion need be hidden from the untrusted system by encryption. Thus, the service function takes three arguments—the current public state, the current private state, and an input—and it yields a record with three fields—the new public state, the new private state, and an output. The service also specifies an initial value for the public state, an initial value for the private state, and an initial value for the set of inputs that are available to be processed by the service. This is described precisely in Section 2.1.

1.2.1 High-Level Specification for Memoir-Basic Semantics

The high-level spec for Memoir-Basic semantics contains four state variables: the current public state, the current private state, the set of inputs that are available to be processed by the service, and the set of outputs that the service has been observed to produce. The latter two warrant some explanation.

The set of available inputs is intended to model the fact that, at any given time, some inputs might not be known to the user that invokes the service. For example, if the service is used to redeem cryptographically signed tokens, the user may not know the complete set of valid tokens. The set of available inputs includes the inputs that, at a given moment, are known by the user and thus available to be processed by the service.

The high-level spec includes a variable for the set of outputs observed from the service, because it is important to show that the low-level specs produce a corresponding set of outputs by their actions. It is not enough to show that the public and private states in the refined low-level specs equal the public and private states of the high-level spec, because this would be insufficient to preclude a low-level spec that returns a different set of outputs than are intended by the semantics.

The high-level spec for Memoir-Basic semantics includes two actions. The main action, *HLAdvanceService*, invokes the service function with arguments of the current public state, the current private state, and an input from the set of available inputs. The output of the service function updates the current public state, the current private state, and the set of outputs observed from the service.

The second action, *HLMakeInputAvailable*, adds an input to the set of available inputs. This action might, for example, model an out-of-band transaction in which the user pays money in exchange for a signed token, thereby enabling the user to submit a request containing that token.

As argued above (§ 1.1.2), this high-level spec is small and simple enough that it should be easy to determine that its semantics are the right ones. In particular, it is readily apparent that the *HLAdvanceService* action provides the state continuity desired for the service module.

1.2.2 Memoir-Basic Low-Level Specification

The Memoir-Basic low-level spec contains six state variables. Three of these variables represent concrete state maintained by a Memoir-Basic implementation: contents of the disk, contents of the RAM, and contents of the NVRAM. The only parts of each storage device we model are those of direct relevance to Memoir. For the NVRAM, this is the history summary and symmetric key that Memoir-Basic stores in the NVRAM. For the RAM, this is the values that are exchanged between Memoir and the untrusted system: the current public state and encrypted private state stored by the untrusted system, and the history summary and authenticator that the untrusted system uses to convince Memoir that the current state is valid. For the disk, this is a copy of the contents of the RAM that are stored on the disk for crash-resilience.

The other three variables represent abstractions, two of which are direct analogues of state variables in the high-level spec: the set of inputs that are available to be processed and the set of outputs that have been observed. The third abstract variable is the set of authenticators that the untrusted system has observed to be returned from Memoir. This set is needed as part of the formalism, because an attacker can attempt to re-use any authenticator it has observed Memoir to produce (c.f. § 1.1.3), so the specification needs to track this set to show that its elements are available to an attacker.

The Memoir-Basic low-leve spec includes seven actions. The only two actions that model the execution of Memoir-Basic code are *LL*1*PerformOperation* and *LL*1*RepeatOperation*. The *LL*1*PerformOperation* action describes both concrete operations performed by the Memoir-Basic implementation and also abstract operations needed for the formalism. The concrete operations include checking the values in the RAM from the untrusted system against values in NVRAM to ensure correctness and currency, invoking the service function with arguments from the RAM and an input from the set of available inputs, and updating the RAM and NVRAM with new values based on the output of the service function. The abstract operations update the sets of observed outputs and observed authenticators.

The *LL*1*RepeatOperation* action behaves similarly to the *LL*1*PerformOperation* action, with two main exceptions: First, instead of checking that the state in the RAM is current, it checks that if the state in the RAM were advanced by the given input from the set of available inputs, the resulting state would be current according to the NVRAM. Second, it does not update the NVRAM. Importantly, the *LL*1*RepeatOperation* action does update the sets of observed outputs and observed authenticators. This may seem odd, because if the Memoir system is functioning correctly, the *LL*1*RepeatOperation* action will not produce an output that it has not previously produced, nor will it produce an authenticator with a meaning other than that of some authenticator it previously produced. However, this is not a property we assume in the definition of the action; it is a property we prove as part of the implementation proof (c.f. the inclusion invariant in § 1.2.4).

A third action, *LL*1*MakeInputAvailable*, is an abstract action that is a direct analogue of the high-level spec's *HLMakeInputAvailable* action. This action adds an input to the set of available inputs but leaves all concrete state unchanged.

There are three actions that model behavior of the untrusted system. The *LL*1*ReadDisk* action reads the state of the disk into the RAM. The *LL*1*WriteDisk* action writes the state of the RAM onto the disk. The *LL*1*Restart* action models the effect of a system restart by trashing the values in the RAM.

The final Memoir-Basic low-level action is *LL*1*CorruptRAM*. This action models an attacker's ability to put nearly arbitrary values in the RAM before invoking Memoir. As described in Section 1.1.3, because the symmetric key stored in the NVRAM is unknown outside the trusted subsystem, the only authenticators the attacker can put in the RAM are (1) those from the set of authenticators that the untrusted system has observed to be returned from Memoir and (2) those the attacker can generate using a symmetric key other than the key stored in the NVRAM.

1.2.3 Refinement of Memoir-Basic State

The refinement describes how to interpret values of state variables in the Memoir-Basic low-level spec as values of state variables in the high-level spec. There are four high-level variables whose values need to be established through the refinement, two of which are trivial: The high-level variables representing the set of available inputs and the set of observed outputs are asserted by the refinement to respectively equal the corresponding sets from the low-level spec. These are abstract variables, and they have identical meanings across the two specs.

Refining the high-level public and private state is more involved. Intuitively, the only concrete value in the low-level spec that determines the current service state is the history summary in the NVRAM. The values in the RAM and the disk are irrelevant, because the untrusted system can set these to any values at any time. So, the refinement needs to express that the high-level values of public and private state are values that correspond (in some strong but as yet ill-defined sense) to the history summary in the NVRAM. The way we express this correspondence is by exploiting the set of observed authenticators. Each authenticator expresses a binding between a history summary (such as the one stored in the NVRAM) and a state hash formed from a public and private state. Thus, the refinement asserts that the high-level variables representing the public and private state have any values whose hash is bound to the history summary currently in the NVRAM by some authenticator in the set of observed authenticators. Although it may not be obvious, this assertion uniquely defines the high-level public and private state. We will prove this uniqueness as part of the implementation proof (c.f. the uniqueness invariant in § 1.2.4).

1.2.4 Memoir-Basic Invariants

The proof that the Memoir-Basic low-level spec implements the high-level spec depends upon three inductive invariants: the *unforgeability invariant*, the *inclusion invariant*, and the *uniqueness invariant*, which we collectively refer to as the *correctness invariants*.

The unforgeability invariant is a somewhat boring invariant. As described above (§ 1.2.2), the definition of the *LL*1*CorruptRAM* action constrains the set of authenticators that can be put into the RAM by an attacker. The unforgeability invariant essentially states that the only authenticator values in the RAM are those that satisfy the constraint imposed by the *LL*1*CorruptRAM* action. In other words, no other actions violate this constraint. Since one of these other actions, *LL*1*ReadDisk*, copies the authenticator from the disk into the RAM, we cannot prove the unforgeability invariant directly. Instead, we first prove the *extended unforgeability invariant*, which applies the authenticator constraint to both the RAM and the disk. The extended unforgeability invariant directly implies the unforgeability invariant.

The inclusion invariant is needed for the implementation proof in the presence of the *LL*1*RepeatOperation* action. This invariant essentially states that (1) the output that *LL*1*RepeatOperation* will produce is already in the set of observed outputs, and (2) the new authenticator that *LL*1*RepeatOperation* will produce authenticates a history state binding that is already being authenticated by some authenticator in the set of observed authenticators. Thus, the *LL*1*RepeatOperation* action will not modify these sets in any semantically important way.

The uniqueness invariant states that the the history summary in the NVRAM is bound to only one public and private state by an authenticator in the set of observed authenticators. This invariant is used in the proof that the initial high-level state is correctly defined, in the proofs that the high-level public and private state is not changed by any low-level action that should not change this state, and in the proof that the low-level *LL*1*PerformOperation* action implements the behavior of the high-level *HLAdvanceService* action.

Just as the proof of the unforgeability invariant relies on the extended unforgeability invariant, the proof of the inclusion invariant and the uniqueness invariant also rely on a supplementary invariant, which we call the *cardinality invariant*. However, unlike the extended unforgeability invariant, the cardinality invariant cannot be proven on its own. Moreover, the inclusion, cardinality, and uniqueness invariants cannot be ordered with respect to each other. The proof of the inclusion invariant inductively depends upon the uniqueness invariant, which in turn depends upon the cardinality invariant, which in turn depends upon the inclusion invariant. We prove these three invariants co-inductively.

The statement of the cardinality invariant relies on a formalism we call the *cardinality* of a hash, which is the count of hashes that went into the production of any value in the domain of the hash function. The hash cardinality is well-defined because of the strong collision-resistance of the hash function that is assumed (§ 1.1.3) by our proof. The hash cardinality of the base hash value is zero; the hash cardinality of any value not output from the hash function is zero; and the hash cardinality of any output from the hash function is one greater than the hash cardinality of the inputs to the hash function.

The cardinality invariant states that the hash cardinality of the history summary bound by any authenticator in the set of observed authenticators is less than or equal to the hash cardinality of the history summary in the NVRAM. The cardinality invariant inductively supports the proof of the uniqueness invariant, because it allows us to prove that when the *LL*1*PerformOperation* action produces a new authenticator, that authenticator binds a history summary that is not bound by any authenticator in the set of observed authenticators, because the new authenticator has a greater hash cardinality than any authenticator in the set. In turn, the uniqueness invariant inductively supports the proof of the inclusion invariant, because it allows us to prove that when the *LL1PerformOperation* action produces a state hash from the public and private state in the RAM, this equals the state hash defined in the inclusion invariant. Completing the cycle, the inclusion invariant inductively supports the proof of the cardinality invariant, because it allows us to prove that the *LL1RepeatOperation* action makes no semantic change to the set of observed authenticators, and thus there is no change to the set of hash cardinalities represented by this set.

1.2.5 Memoir-Basic Correctness

To prove that the Memoir-Basic low-level spec implements the high-level spec, we prove (1) that the initial state of the low-level spec, under refinement, satisfies the initial state of the high-level spec, and (2) the next-state predicate of the low-level spec, under refinement, satisfies the next-state predicate of the high-level spec. Moreover, the proof of the next-state predicate includes sub-proofs for the following eight implications:

$\textsc{unchanged}\ LL1Vars \Rightarrow \textsc{unchanged}\ HLVars$
$LL1MakeInputAvailable \Rightarrow HLMakeInputAvailable$
$LL1PerformOperation \Rightarrow HLAdvanceService$
$LL1RepeatOperation \Rightarrow \textsc{unchanged}\ HLVars$
$LL1Restart \Rightarrow \textsc{unchanged}\ HLVars$
$LL1ReadDisk \Rightarrow \textsc{unchanged}\ HLVars$
$LL1WriteDisk \Rightarrow \textsc{unchanged}\ HLVars$
$LL1CorruptRAM \Rightarrow \textsc{unchanged}\ HLVars$

In other words:

- A Memoir-Basic stuttering step maps to a high-level stuttering step.
- A Memoir-Basic *LL1MakeInputAvailable* action maps to a high-level *HLMakeInputAvailable* action.
- A Memoir-Basic *LL1PerformOperation* action maps to a high-level *HLAdvanceService* action.
- All other Memoir-Basic actions map to high-level stuttering steps.

The proofs of high-level stuttering all exploit a lemma called the *non-advancement lemma*. This lemma states that, if there is no change to the NVRAM or to the authentication status of any history state binding, then there is no change to the high-level public and private state defined by the refinement. Employing this lemma is completely straightforward for a low-level stuttering step and for the *LL1Restart*, *Ll1ReadDisk*, *LL1WriteDisk*, and *LL1CorruptRAM* actions. For the *LL1RepeatOperation* action, this lemma is usable because the inclusion invariant guarantees that *LL1RepeatOperation* does not change the authentication status of any history state binding.

The proof for the *LL1MakeInputAvailable* action is straightforward. The set of available inputs corresponds directly across the two specs, and the non-advancement lemma shows that the high-level state does not change.

The proof for *LL1PerformOperation* uses the uniqueness invariant twice: First, it is used to show that the public and private state in the arguments to the service correspond to the refined high-level state. Second, it is used to show that the service results in a public and private state that corresponds to the refined high-level primed state. Thus, the service processes the same inputs and produces the same outputs as the service in the *HLAdvanceService* action in the high-level spec.

The following table shows which invariants are needed for which action's proof:

Predicate	unforgeability invariant	inclusion invariant	uniqueness invariant
$LL1Init$	-	-	✓
$\textsc{unchanged}\ LL1Vars$	-	-	✓
$LL1MakeInputAvailable$	-	-	✓
$LL1PerformOperation$	✓	-	✓
$LL1RepeatOperation$	✓	✓	✓
$LL1Restart$	-	-	✓
$LL1ReadDisk$	-	-	✓
$LL1WriteDisk$	-	-	✓
$LL1CorruptRAM$	-	-	✓

1.3 Memoir-Opt

The Memoir-Opt system has a different high-level specification than the Memoir-Basic system. In particular, there are actions in the Memoir-Opt system that enable a malicious user of the system to permanently kill the system. Although we cannot prevent the user from killing the system, we wish to ensure that the only undesirable behavior that can happen is the death of the system, meaning that it stops processing inputs and produces no new outputs. Therefore, we modify the high-level spec to add this behavior to the system semantics.

Our approach to proving the correctness of Memoir-Opt is to prove that, under refinement, the Memoir-Opt spec satisfies the Memoir-Basic spec, which transitively implies that it satisfies the high-level spec. However, because Memoir-Opt includes actions that can kill the system, whereas Memoir-Basic does not, it is not possible for the Memoir-Opt spec to satisfy the Memoir-Basic spec we have described so far. Therefore, we modify the Memoir-Basic spec to include an additional action that does not represent any realistic action in a direct implementation of the Memoir-Basic spec. We will prove that this new action in the Memoir-Basic spec maps to a system death in the high-level spec. Then, we will prove that certain actions in the Memoir-Opt spec, under certain conditions, will map to this new action in the Memoir-Basic spec.

1.3.1 Modifications to High-Level Specification for Memoir-Opt Semantics

To support refinement from the Memoir-Opt spec, we modify the high-level spec to add an additional action, which in turn requires adding an additional state variable.

The new state variable is simply a boolean that indicates whether the system is alive. We modify the initial-state predicate to indicate that this variable is true in the initial system state. We also add an enablement condition to the existing high-level *HLAdvanceService* action to require this variable to be true; in other words, the system must be alive for the *HLAdvanceService* action to occur.

The new action we add is *HLDie*, which does nothing other than set the new state variable to false. Once the variable becomes false, there is no action that will set it back to true.

1.3.2 Modifications to Memoir-Basic Specification for Memoir-Opt Semantics

To support refinement from the Memoir-Opt spec, we modify the Memoir-Basic spec to add an additional action. This new action, *LL*1*RestrictedCorruption*, does not model any realistic action in a direct implementation of the Memoir-Basic spec. In particular, this action corrupts the history summary stored in the NVRAM, but the TPM prevents any code other than Memoir from writing to the NVRAM.

The purpose of this action is to model the effect on the Memoir-Basic spec that refines from the Memoir-Opt spec when the SPCR in Memoir-Opt is corrupted or inappropriately reset. We need the *LL*1*RestrictedCorruption* action to be strong enough to enable refinement from actions the Memoir-Opt spec but weak enough to enable refinement to the *HLDie* action in the high-level spec.

We therefore impose two constraints on the corrupted history summary value in the NVRAM. First, to ensure that the *CardinalityInvariant* and *UniquenessInvariant* continue to hold, no authenticator in the set of observed authenticators may validate a history state binding that binds the NVRAM's history summary to any state hash. Second, to ensure that the *InclusionInvariant* continues to hold, no authenticator in the set of observed authenticators may validate a history state binding that binds any predecessor of the the NVRAM's history summary to any state hash.

The *LL*1*RestrictedCorruption* action may also corrupt the state of the RAM in the exact same way the *LL*1*Restart* action corrupts the RAM, or it may leave the RAM unchanged. Both alternatives are necessary because two different actions in the Memoir-Opt spec refine to the *LL*1*RestrictedCorruption* action, and although they have the same effect on the NVRAM, they have different effects on the RAM.

1.3.3 Memoir-Opt Low-Level Specification

The Memoir-Opt low-level spec contains seven state variables, three of which represent the same abstractions represented in the Memoir-Basic low-level spec: the set of available inputs, the set of observed outputs, and the set of observed authenticators. The other four state variables represent concrete state maintained by a Memoir-Opt implementation: disk, RAM, NVRAM, and SPCR. The disk and RAM variables are direct analogues of the disk and RAM variables in Memoir-Basic.

The NVRAM variable contains a history summary and symmetric key, just like the Memoir-Basic NVRAM. However, it also stores two additional fields: a hash barrier secret and flag indicating whether an extension is in progress. The SPCR variable, unsurprisingly, models the the SPCR.

The Memoir-Opt low-level spec includes nine actions. Four of these actions are semantically identical to corresponding actions in the Memoir-Basic spec: $LL2MakeInputAvailable$, $LL2ReadDisk$, $LL2WriteDisk$, and $LL2CorruptRAM$. Three other actions, although not identical, are semantically analogous to actions in the Memoir-Basic spec: $LL2PerformOperation$, $LL2RepeatOperation$, and $LL2Restart$. The first two of these action differ from their Memoir-Basic counterparts as described in the Memoir paper.[8] The third action, $LL2Restart$, is different only in that it additionally resets the state of the SPCR.

The remaining two actions have no counterparts in the Memoir-Basic spec. The $LL2TakeCheckpoint$ action takes a checkpoint, updating the state of the NVRAM to include the history summary information from the SPCR. The $LL2CorruptSPCR$ action models an attacker's ability to modify the contents of the SPCR by extending it with a nearly arbitrary value. As described in Section 1.1.3, the precise specification of $LL2CorruptSPCR$ ensures that (1) the SPCR can only be modified by extending its hash chain, and (2) because the hash barrier stored in the NVRAM is unknown outside the trusted subsystem, the value that extends the PCR cannot incorporate the hash barrier.

1.3.4 Refinement of Memoir-Opt State

The refinement describes how to interpret values of state variables in the Memoir-Opt low-level spec as values of state variables in the Memoir-Basic low-level spec. There are three cases for how this interpretation is handled.

The first and simplest case is variables that are directly equal between the two specs. This includes the set of available inputs, the set of observed outputs, and some of the fields of the disk, the RAM, and the NVRAM. For the disk and RAM, the particular fields that are equal across the two specs are the public state and the encrypted private state. For the NVRAM, the symmetric key is equal across the two specs.

The second case is variables that directly "match" across the two specs. This includes the set of observed authenticators and the fields of the disk and RAM that are not (as described above) directly equal, namely the authenticator and history summary. Two authenticators match if they are MACs of history state bindings that bind matching history summaries to equal state hashes. Two history summaries match if they both equal the respective initial history summaries for the two specs or (recursively) if they are both successors (with the same input) of matching history summaries.

The third and most involved case is the history summary in the Memoir-Basic NVRAM, which is refined to match the logical value of the history summary defined by the Memoir-Opt NVRAM and SPCR. The logical value of the anchor is the anchor value in the NVRAM, but the logical extension is the value in the SPCR only if the NVRAM indicates that an extension is in progress; otherwise, the logical extension equals the base hash value. The reason for this is that the $LL2TakeCheckpoint$ action clears the flag that indicates whether an extension is in progress, but it does not reset the SPCR to the base hash value. Therefore, between an $LL2TakeCheckpoint$ action and an $LL2Restart$ action, the logical extension is really the base hash value, even though the SPCR has not yet been reset.

1.3.5 Memoir-Opt Correctness

Since we modified the Memoir-Basic low-level spec by adding a new action, we need to state and prove the mapping of this action to the high-level spec. Specifically, in the Memoir-Basic implementation proof, we add an additional sub-proof to the proof of the next-state predicate for the following implication:

$LL1RestrictedCorruption \Rightarrow HLDie$

Then, to prove that the Memoir-Opt low-level spec implements the Memoir-Basic low-level spec, we prove (1) that the initial state of the Memoir-Opt spec, under refinement, satisfies the initial state of the Memoir-Basic spec, and (2) the next-state predicate of the Memoir-Opt spec, under refinement, satisfies the next-state predicate of the Memoir-Basic spec. Moreover, the proof of the next-state predicate includes sub-proofs for the following ten implications:

$\text{UNCHANGED } LL2Vars \Rightarrow \text{UNCHANGED } LL1Vars$
$LL2MakeInputAvailable \Rightarrow LL1MakeInputAvailable$
$LL2PerformOperation \Rightarrow LL1PerformOperation$
$LL2RepeatOperation \Rightarrow LL1RepeatOperation$
$LL2TakeCheckpoint \Rightarrow \text{UNCHANGED } LL1Vars$
$LL2Restart \Rightarrow$
$\quad \text{IF } LL2NVRAM.extensionInProgress$

THEN

$\quad LL1RestrictedCorruption$

ELSE

$\quad LL1Restart$

$LL2ReadDisk \Rightarrow LL1ReadDisk$

$LL2WriteDisk \Rightarrow LL1WriteDisk$

$LL2CorruptRAM \Rightarrow LL1CorruptRAM$

$LL2CorruptSPCR \Rightarrow$

IF $LL2NVRAM.extensionInProgress$

THEN

$\quad LL1RestrictedCorruption$

ELSE

$\quad$ UNCHANGED $LL1Vars$

In other words:

- A Memoir-Opt stuttering step maps to a Memoir-Basic stuttering step.
- Six Memoir-Opt actions directly map to analogous Memoir-Basic actions; these are *LL2MakeInputAvailable*, *LL2PerformOperation*, *LL2RepeatOperation*, *LL2ReadDisk*, *LL2WriteDisk*, and *LL2CorruptRAM*.
- A Memoir-Opt *LL2TakeCheckpoint* action maps to a Memoir-Basic stuttering step.
- A Memoir-Opt *LL2Restart* action maps either to an *LL1RestrictedCorruption* action or to an *LL1Restart* action depending on whether an extension is in progress.
- A Memoir-Opt *LL2CorruptSPCR* action maps either to an *LL1RestrictedCorruption* action or to a Memoir-Basic stuttering step depending on whether an extension is in progress.

In the above list, the final two bullet points merit explanation. It might seem that an *LL2Restart* action should map directly to an *LL1Restart* action, and this is the case under normal operation. In particular, since Memoir-Opt should always perform an *LL2TakeCheckpoint* action immediately prior to restarting, and since the *LL2TakeCheckpoint* action clears the extension-in-progress flag, a subsequent *LL2Restart* action (with no intervening *LL2PerformOperation* action) will map to *LL1Restart*. However, a malicious user can force the system to restart without first taking a checkpoint. If this happens when an extension is in progress, the Memoir system will die. We prove this by the transitive implication:

$$LL2NVRAM.extensionInProgress \land LL2Restart \Rightarrow LL1RestrictedCorruption \Rightarrow HLDie$$

Irrespective of whether an extension is in progress, the *LL2Restart* action corrupts the state of the RAM in the exact same way the *LL1Restart* action corrupts the RAM. This is not a problem for the above implication, because we have specified the *LL1RestrictedCorruption* action to allow corruption of the RAM in this exact same manner.

Complementary reasoning applies to the mapping of *LL2CorruptSPCR*. It might seem that this action should map directly to an *LL1RestrictedCorruption* action, since the SPCR holds important data about the service state. However, when the flag in the NVRAM indicates that an extension is not in progress, the state of the SPCR is supposed to equal the base hash value. Therefore, even if the SPCR is corrupted by an *LL2CorruptSPCR* action, the SPCR can be restored to its proper value by an *LL2Restart* action, after which normal operation can resume. Thus, when an extension is not in progress, the *LL2Restart* action does not cause the system to die, which we prove with the following transitive implication:

$$\neg LL2NVRAM.extensionInProgress \land LL2CorruptSPCR \Rightarrow \text{UNCHANGED } LL1Vars \Rightarrow \text{UNCHANGED } HLVars$$

Note that if we were specifying liveness as well as safety, this implication would not hold, because the states before and after an *LL2CorruptSPCR* action differ in their liveness, insofar as an *LL2PerformOperation* action can occur beforehand but not afterward, unless it is preceded by an *LL2Restart* action.

1.4 Organization

The remainder of this tech report includes the following items:

High-level spec: There is a single high-level spec that defines the semantics of the Memoir system. It includes both the basic semantics of the Memoir-Basic implementation and also the additional semantics of the Memoir-Opt implementation.

Low-level primitives: The low-level specifications make use of several primitives, namely a hash function, MAC functions, and symmetric cryptography. These functions are specified by undefined operators, along with explicit assumptions about the guarantees made by the operators.

Memoir-Basic low-level spec: The Memoir-Basic spec describes how a Memoir-Basic implementation behaves, in terms of input, output, and operations on the disk, RAM, and NVRAM. This spec also includes an additional action (which is not part of a a Memoir-Basic implementation) that supports refinement from the Memoir-Opt specification.

Memoir-Opt low-level spec: The Memoir-Opt spec describes how a Memoir-Opt implementation behaves, in terms of input, output, and operations on the disk, RAM, NVRAM, and SPCR.

Refinements: There are two refinements. One describes the mapping of Memoir-Basic state to high-level state. The other describes the mapping of Memoir-Opt state to Memoir-Basic state.

Invariants: There are five invariants maintained by the Memoir-Basic specification, three of which are needed by the proof that the Memoir-Basic spec satisfies the high-level spec. There are no invariants needed for the proof that the Memoir-Opt spec satisfies the Memoir-Basic spec.

Type-safety theorems: TLA+ is an untyped language, so we state and prove the types of all variables maintained by each of the three specs.

Invariance theorems: The invariants are proven using a temporal inductive proof rule called *Inv1*. To employ this rule, we prove that each invariant is satisfied in the initial system state, and we prove that each valid action preserves the invariant.

Implementation theorems: There are two implementation theorems: The first states that Memoir-Basic implements the high-level spec, and the second states that Memoir-Opt implements Memoir-Basic. Each implementation is proven using a temporal inductive proof rule called *StepSimulation*. To employ this rule, we prove that the initial state of each lower-level spec, under refinement, satisfies the initial state of the corresponding higher-level spec, and that each lower-level action corresponds to a higher-level action.

Ancillary Lemmmas: There are quite a few lemmas that support the invariance proofs and/or the implementation proofs.

These items are partitioned into TLA+ organizational structures called *modules*, which group related items together. There are 21 modules in this set of specifications and proofs. Within Sections 2–4, each subsection corresponds to one module.

1.4.1 Organization of TLA+ Modules

Multiple modules are combined by the process of *extension**. Each module can extend the declarations and definitions of one or more other modules. The Memoir modules are organized into a linear chain, wherein each of the following modules extends the one before it:

- MemoirCommon—declarations common to high- and low-level specs
- MemoirHLSpecification—specification of the high-level system (semantics)
- MemoirHLTypeSafety—proof of type safety of the high-level spec
- MemoirLLPrimitives—primitives used by the low-level systems
- MemoirLL1Specification—specification of the Memoir-Basic system
- MemoirLL1Refinement—refinement 1: mapping Memoir-Basic state to high-level state
- MemoirLL1TypeLemmas—proofs of lemmas relating to types in the Memoir-Basic Spec
- MemoirLL1TypeSafety—proof of type safety of the Memoir-Basic spec
- MemoirLL1CorrectnessInvariants—invariants needed to prove Memoir-Basic implementation
- MemoirLL1SupplementalInvariants—invariants needed to prove Memoir-Basic invariance
- MemoirLL1InvarianceLemmas—proofs of lemmas that support Memoir-Basic invariance proofs

*This use of the term 'extension' is completely unrelated to the 'extension' performed on the SPCR. The name collision is unfortunate but unavoidable, since both uses predate our work.

- MemoirLL1UnforgeabilityInvariance—proof of unforgeability invariance in Memoir-Basic
- MemoirLL1InclCardUniqInvariance—proof of inclusion, cardinality, and uniqueness co-invariance in Memoir-Basic
- MemoirLL1Implementation—proof that Memoir-Basic spec implements high-level spec
- MemoirLL2Specification—specification of the Memoir-Opt system
- MemoirLL2Refinement—refinement 2: mapping Memoir-Opt state to Memoir-Basic state
- MemoirLL2TypeLemmas—proofs of lemmas relating to types in the Memoir-Opt spec
- MemoirLL2TypeSafety—proof of type safety of the Memoir-Opt spec
- MemoirLL2RefinementLemmas—proofs of lemmas relating to the Memoir-Opt refinement
- MemoirLL2ImplementationLemmas—proofs of lemmas relating to the Memoir-Opt implementation
- MemoirLL2Implementation—proof that Memoir-Opt spec implements Memoir-Basic spec

This organization largely reflects the order in which the modules were developed. More importantly, the organization places each proof roughly as early in the chain as possible, so that only the items it depends on come before it.

However, within this document, we present the modules in a different order that is more conducive to reading. First, we present the modules pertaining to the specification of the system's behavior and its implementation (Section 2, beginning on page 18). Second, we present the modules pertaining to the refinement of one spec to another, as well as modules describing invariants maintained by the specs (Section 3, beginning on page 40). Third, we present the modules that contain proofs (Section 4, beginning on page 51).

1.4.2 Index of Declarations

Following is a complete list of all declarations in this set of formal specs and proofs, along with the page on which the declaration can be found. The declarations are partitioned into those of constants, variables, definitions, assumptions, and theorems.

Constants

Variables

Definitions

Assumptions

Theorems

2. SPECIFICATIONS

This section presents TLA+ modules pertaining to the specification of the system's behavior and its implementation. This includes common declarations, definitions of low-level primitives, and specifications for the high-level spec and both low-level specs.

As a guide to understanding the impact of the actions in each spec, the following tables show which state variables are read and/or written by each action. To keep these tables from being useless, we employ definitions of "read" and "written" that are slightly non-obvious with respect to the formal specification. In particular, we ignore the fact that the UNCHANGED predicate both "reads" and "writes" a variable, insofar as it specifies that the primed state of the variable equals the unprimed state of that variable.

Action	HL Alive	HLAvailable Inputs	HLObserved Outputs	HLPublic State	HLPrivate State
HLMakeInputAvailable	-	R/W	-	-	-
HLAdvanceService	R	R	R/W	R/W	R/W
HLDie	W	-	-	W	W

Action	LL1Available Inputs	LL1Observed Outputs	LL1Observed Authenticators	LL1 Disk	LL1 RAM	LL1 NVRAM
LL1MakeInputAvailable	R/W	-	-	-	-	-
LL1PerformOperation	R	R/W	R/W	-	R/W	R/W
LL1RepeatOperation	R	R/W	R/W	-	R/W	R
LL1Restart	-	-	-	-	W	R
LL1ReadDisk	-	-	-	R	W	-
LL1WriteDisk	-	-	-	W	R	-
LL1CorruptRAM	-	-	-	-	W	R
LL1RestrictedCorruption	-	-	-	-	-	R/W

Action	LL2Available Inputs	LL2Observed Outputs	LL2Observed Authenticators	LL2 Disk	LL2 RAM	LL2 NVRAM	LL2 SPCR
LL2MakeInputAvailable	R/W	-	-	-	-	-	-
LL2PerformOperation	R	R/W	R/W	-	R/W	R/W	R/W
LL2RepeatOperation	R	R/W	R/W	-	R/W	R	-
LL2TakeCheckpoint	-	-	-	-	-	R/W	R
LL2Restart	-	-	-	-	W	R	R/W
LL2ReadDisk	-	-	-	R	W	-	-
LL2WriteDisk	-	-	-	W	R	-	-
LL2CorruptRAM	-	-	-	-	W	R	-
LL2CorruptSPCR	-	-	-	-	-	R	R/W

2.1 Declarations Common to High- and Low-Level Specs

MODULE *MemoirCommon*

This module defines some basic constants used by both the high-level and low-level specs.

A developer that wishes to use Memoir is expected to provide a service implementation that (1) operates on some application-specific input, (2) produces some application-specific output, and (3) maintains some application-specific public and private state. The developer also specifies an initial public and private state for the service. The service is assumed to be type-safe.

The one non-obvious aspect of this module is the constant *InitialAvailableInputs*, which will be explained in the comments relating to the high-level spec.

EXTENDS *TLAPS*

CONSTANT *InputType*
CONSTANT *OutputType*
CONSTANT *PublicStateType*
CONSTANT *PrivateStateType*

$$
\begin{aligned}
&\mathit{ServiceResultType} \triangleq \\
&\quad [\ \ \mathit{newPublicState} : \mathit{PublicStateType}, \\
&\quad\quad \mathit{newPrivateState} : \mathit{PrivateStateType}, \\
&\quad\quad \mathit{output} : \mathit{OutputType} \\
&\quad]
\end{aligned}
$$

CONSTANT $\mathit{Service}(\_, \_, \_)$

ASSUME $\mathit{ServiceTypeSafe} \triangleq$

$$
\begin{aligned}
&\forall\, \mathit{input} \in \mathit{InputType},\ \mathit{publicState} \in \mathit{PublicStateType},\ \mathit{privateState} \in \mathit{PrivateStateType} : \\
&\quad \mathit{Service}(\mathit{publicState}, \mathit{privateState}, \mathit{input}) \in \mathit{ServiceResultType}
\end{aligned}
$$

CONSTANT *InitialAvailableInputs*
CONSTANT *InitialPublicState*
CONSTANT *InitialPrivateState*
CONSTANT *DeadPublicState*
CONSTANT *DeadPrivateState*

ASSUME $\mathit{ConstantsTypeSafe} \triangleq$

$$
\begin{aligned}
&\land\ \ \mathit{InitialAvailableInputs} \subseteq \mathit{InputType} \\
&\land\ \ \mathit{InitialPublicState} \in \mathit{PublicStateType} \\
&\land\ \ \mathit{InitialPrivateState} \in \mathit{PrivateStateType} \\
&\land\ \ \mathit{DeadPublicState} \in \mathit{PublicStateType} \\
&\land\ \ \mathit{DeadPrivateState} \in \mathit{PrivateStateType}
\end{aligned}
$$

2.2 Specification of the High-Level System (Semantics)

MODULE *MemoirHLSpecification*

This module defines the high-level behavior of Memoir. There are three actions:
HLMakeInputAvailable
HLAdvanceService
HLDie

EXTENDS *MemoirCommon*

VARIABLE *HLAlive*
VARIABLE *HLAvailableInputs*
VARIABLE *HLObservedOutputs*
VARIABLE *HLPublicState*
VARIABLE *HLPrivateState*

$$
\begin{aligned}
&HLTypeInvariant \triangleq \\
&\quad \land\ HLAlive \in \text{BOOLEAN} \\
&\quad \land\ HLAvailableInputs \subseteq InputType \\
&\quad \land\ HLObservedOutputs \subseteq OutputType \\
&\quad \land\ HLPublicState \in PublicStateType \\
&\quad \land\ HLPrivateState \in PrivateStateType
\end{aligned}
$$

$$HLVars \triangleq \langle HLAlive, HLAvailableInputs, HLObservedOutputs, HLPublicState, HLPrivateState \rangle$$

The *HLAdvanceService* action is not allowed to take just any input from *InputType*. It may only take an input from the set *HLAvailableInputs*. This models the fact that some inputs might not be known to the user that invokes the service. For example, the service might be used to redeem cryptographically signed tokens, and the user does not initially know the complete set of valid tokens. The user might, for example, have to pay money to retrieve a token from a server, and when the user does so, this corresponds to the action *HLMakeInputAvailable*, which puts the input that includes this token into the set of *HLAvailableInputs*.

$$
\begin{aligned}
&HLMakeInputAvailable \triangleq \\
&\quad \exists\, input \in InputType : \\
&\qquad \land\ input \notin HLAvailableInputs \\
&\qquad \land\ HLAvailableInputs' = HLAvailableInputs \cup \{input\} \\
&\qquad \land\ \text{UNCHANGED } HLObservedOutputs \\
&\qquad \land\ \text{UNCHANGED } HLAlive \\
&\qquad \land\ \text{UNCHANGED } HLPublicState \\
&\qquad \land\ \text{UNCHANGED } HLPrivateState
\end{aligned}
$$

The high-level behavior is a service. There is one main action, which is *HLAdvanceService*. This action takes some input, invokes the developer-supplied service with this input and the current public and private state, updates the public and private state accordingly, and adds the output to the set of observed outputs.

$$
\begin{aligned}
&HLAdvanceService \triangleq \\
&\quad \exists\, input \in HLAvailableInputs : \\
&\qquad \text{LET} \\
&\qquad\quad hlSResult \triangleq Service(HLPublicState, HLPrivateState, input) \\
&\qquad \text{IN} \\
&\qquad\quad \land\ HLAlive = \text{TRUE} \\
&\qquad\quad \land\ HLPublicState' = hlSResult.newPublicState \\
&\qquad\quad \land\ HLPrivateState' = hlSResult.newPrivateState \\
&\qquad\quad \land\ HLObservedOutputs' = HLObservedOutputs \cup \{hlSResult.output\}
\end{aligned}
$$

$\wedge$ UNCHANGED $HLAvailableInputs$
$\wedge$ UNCHANGED $HLAlive$

The high-level spec includes the *HLDie* action, which kills the system. This is necessary because, in the low-level specs, it is possible for an adversary to perform an action that causes the system to no longer function. Therefore, we must admit a corresponding action in the high-level spec. Importantly, the *HLDie* action does not change the set of observed outputs, so it cannot be used to trick the system into providing an output it would not otherwise be willing to provide.

$HLDie \triangleq$
$\wedge$ $HLAlive' = $ FALSE
$\wedge$ UNCHANGED $HLAvailableInputs$
$\wedge$ UNCHANGED $HLObservedOutputs$
$\wedge$ $HLPublicState' = DeadPublicState$
$\wedge$ $HLPrivateState' = DeadPrivateState$

$HLInit \triangleq$
$\wedge$ $HLAlive = $ TRUE
$\wedge$ $HLAvailableInputs = InitialAvailableInputs$
$\wedge$ $HLObservedOutputs = \{\}$
$\wedge$ $HLPublicState = InitialPublicState$
$\wedge$ $HLPrivateState = InitialPrivateState$

$HLNext \triangleq$
$\vee$ $HLMakeInputAvailable$
$\vee$ $HLAdvanceService$
$\vee$ $HLDie$

$HLSpec \triangleq HLInit \wedge \Box[HLNext]_{HLVars}$

2.3 Primitives Used by the Low-Level Systems

MODULE *MemoirLLPrimitives*

This module defines primitives that are used by the low-level specs. The primitives include a hash function, *MAC* functions, and symmetric crypto functions, along with their associated types. The module also asserts assumptions about the properties of these functions.

EXTENDS *MemoirHLTypeSafety*

The low-level specs make use of three primitives: a secure hash, a *MAC* (message authentication code), and symmetric cryptography.

CONSTANT $HashType$
CONSTANT $MACType$
CONSTANT $SymmetricKeyType$
CONSTANT $PrivateStateEncType$

CONSTANT $Hash(\_, \_)$
CONSTANT $GenerateMAC(\_, \_)$
CONSTANT $ValidateMAC(\_, \_, \_)$
CONSTANT $SymmetricEncrypt(\_, \_)$
CONSTANT $SymmetricDecrypt(\_, \_)$

The hash function has a somewhat strange signature. It accepts two arguments, rather than one. The reason for this is so that we can construct hash chains. Alternatively, we could have written the spec with a conventional single-argument hash function and a two-argument concatenation function, but this would have added complexity for no real benefit.

We assume a base hash value, which in a real implementation, might just be the value zero.

CONSTANT $BaseHashValue$

The domain of the hash function is hashes, inputs, public states, and encrypted private states. These are the only types we need to hash.

$$HashDomain \triangleq \text{UNION } \{HashType, InputType, PublicStateType, PrivateStateEncType\}$$

The base hash value is a valid hash value, and it cannot be produced by hashing any other value.

ASSUME $BaseHashValueTypeSafe \triangleq BaseHashValue \in HashType$

ASSUME $BaseHashValueUnique \triangleq$
$$\forall\, hashInput1,\ hashInput2 \in HashDomain : Hash(hashInput1,\ hashInput2) \neq BaseHashValue$$

The hash function is assumed to be type-safe and collision-resistant. In this spec, we define collision resistance in a very strong sense, namely that there are no different inputs that will hash to the same output. Although a real implementation of a hash function cannot satisfy this, cryptographically secure hash functions are expected to practically satisfy such a condition.

ASSUME $HashTypeSafe \triangleq$
$$\forall\, hashInput1,\ hashInput2 \in HashDomain : Hash(hashInput1,\ hashInput2) \in HashType$$

$$\text{ASSUME } HashCollisionResistant \triangleq \forall hashInput1a, hashInput2a, hashInput1b, hashInput2b \in HashDomain : Hash(hashInput1a, hashInput2a) = Hash(hashInput1b, hashInput2b) \Rightarrow \land\ hashInput1a = hashInput1b \land\ hashInput2a = hashInput2b$$

The MAC functions are assumed to be type-safe, complete, consistent, unforgeable, and collision-resistant.

$$\text{ASSUME } GenerateMACTypeSafe \triangleq \forall key \in SymmetricKeyType, hash \in HashType : GenerateMAC(key, hash) \in MACType$$

$$\text{ASSUME } ValidateMACTypeSafe \triangleq \forall key \in SymmetricKeyType, hash \in HashType, mac \in MACType : ValidateMAC(key, hash, mac) \in \text{BOOLEAN}$$

$$\text{ASSUME } MACComplete \triangleq \forall key \in SymmetricKeyType, hash \in HashType : ValidateMAC(key, hash, GenerateMAC(key, hash)) = \text{TRUE}$$

$$\text{ASSUME } MACConsistent \triangleq \forall key \in SymmetricKeyType, hash \in HashType, mac \in MACType : ValidateMAC(key, hash, mac) \Rightarrow mac = GenerateMAC(key, hash)$$

$$\text{ASSUME } MACUnforgeable \triangleq \forall key1, key2 \in SymmetricKeyType, hash1, hash2 \in HashType : ValidateMAC(key1, hash1, GenerateMAC(key2, hash2)) \Rightarrow key1 = key2$$

$$\text{ASSUME } MACCollisionResistant \triangleq \forall key1, key2 \in SymmetricKeyType, hash1, hash2 \in HashType : ValidateMAC(key1, hash1, GenerateMAC(key2, hash2)) \Rightarrow hash1 = hash2$$

The symmetric-crypto functions are assumed to be type-safe. They are also assumed to be correct, meaning that decryption is the inverse of encryption, given the same crypto key.

$$\text{ASSUME } SymmetricEncryptionTypeSafe \triangleq \forall key \in SymmetricKeyType, privateState \in PrivateStateType : SymmetricEncrypt(key, privateState) \in PrivateStateEncType$$

$$\text{ASSUME } SymmetricDecryptionTypeSafe \triangleq \forall key \in SymmetricKeyType, privateStateEnc \in PrivateStateEncType : SymmetricDecrypt(key, privateStateEnc) \in PrivateStateType$$

$$\text{ASSUME } SymmetricCryptoCorrect \triangleq \forall key \in SymmetricKeyType, privateState \in PrivateStateType : SymmetricDecrypt(key, SymmetricEncrypt(key, privateState)) = privateState$$

2.4 Specification of the Memoir-Basic System

MODULE *MemoirLL1Specification*

This module defines the low-level specification of Memoir-Basic.

There are eight actions:
$LL1MakeInputAvailable$
$LL1PerformOperation$
$LL1RepeatOperation$
$LL1Restart$
$LL1ReadDisk$
$LL1WriteDisk$
$LL1CorruptRAM$
$LL1RestrictedCorruption$

EXTENDS *MemoirLLPrimitives*

The disk and RAM are untrusted. They each can store the service's public state, encrypted private state, a history summary (a chained hash of all inputs that the service has processed), and an authenticator (a MAC that binds the history summary to the public and private state).

$$
\begin{aligned}
&LL1UntrustedStorageType \triangleq [\\
&\quad publicState : PublicStateType,\\
&\quad privateStateEnc : PrivateStateEncType,\\
&\quad historySummary : HashType,\\
&\quad authenticator : MACType]
\end{aligned}
$$

The $NVRAM$ is trusted, since the TPM guarantees that it can only be read or written by the code that implements Memoir. The $NVRAM$ stores the current history summary and the symmetric key that is used (1) to encrypt the private state and (2) to MAC the history summary and service state into an authenticator.

$$
\begin{aligned}
&LL1TrustedStorageType \triangleq [\\
&\quad historySummary : HashType,\\
&\quad symmetricKey : SymmetricKeyType]
\end{aligned}
$$

$LL1AvailableInputs$ and $LL1ObservedOutputs$ are abstract variables that do not directly represent part of the implementation. They correspond to the $HLAvailableInputs$ and $HLObservedOutputs$ variables in the high-level spec.

VARIABLE $LL1AvailableInputs$
VARIABLE $LL1ObservedOutputs$

The $LL1ObservedAuthenticators$ variable is also abstract. It records the set of authenticators that the user has seen from Memoir. A malicious user can attempt to use these state authenticators in a replay attack against Memoir.

VARIABLE $LL1ObservedAuthenticators$

The $LL1Disk$, $LL1RAM$, and $LL1NVRAM$ variables represent concrete state maintained by the Memoir-Basic implementation.

VARIABLE $LL1Disk$
VARIABLE $LL1RAM$
VARIABLE $LL1NVRAM$

$$
\begin{aligned}
&LL1TypeInvariant \triangleq\\
&\quad \land\ LL1AvailableInputs \subseteq InputType\\
&\quad \land\ LL1ObservedOutputs \subseteq OutputType\\
&\quad \land\ LL1ObservedAuthenticators \subseteq MACType
\end{aligned}
$$

$\land$ $LL1Disk \in LL1UntrustedStorageType$
$\land$ $LL1RAM \in LL1UntrustedStorageType$
$\land$ $LL1NVRAM \in LL1TrustedStorageType$

$LL1Vars \triangleq \langle$
$LL1AvailableInputs,$
$LL1ObservedOutputs,$
$LL1ObservedAuthenticators,$
$LL1Disk,$
$LL1RAM,$
$LL1NVRAM\rangle$

The *LL1MakeInputAvailable* action is a direct analog of the *HLMakeInputAvailable* action in the high-level spec.

$LL1MakeInputAvailable \triangleq$
$\exists\, input \in InputType :$
$\land$ $input \notin LL1AvailableInputs$
$\land$ $LL1AvailableInputs' = LL1AvailableInputs \cup \{input\}$
$\land$ UNCHANGED $LL1Disk$
$\land$ UNCHANGED $LL1RAM$
$\land$ UNCHANGED $LL1NVRAM$
$\land$ UNCHANGED $LL1ObservedOutputs$
$\land$ UNCHANGED $LL1ObservedAuthenticators$

The *LL1PerformOperation* action is invoked by the user to perform a service operation. It is intended to provide the semantics of the *HLAdvanceService* action in the high-level spec.

$LL1PerformOperation \triangleq$
$\exists\, input \in LL1AvailableInputs :$
LET
$stateHash \triangleq Hash(LL1RAM.publicState, LL1RAM.privateStateEnc)$
$historyStateBinding \triangleq Hash(LL1RAM.historySummary, stateHash)$
$privateState \triangleq SymmetricDecrypt(LL1NVRAM.symmetricKey, LL1RAM.privateStateEnc)$
$sResult \triangleq Service(LL1RAM.publicState, privateState, input)$
$newPrivateStateEnc \triangleq$
$SymmetricEncrypt(LL1NVRAM.symmetricKey, sResult.newPrivateState)$
$newHistorySummary \triangleq Hash(LL1NVRAM.historySummary, input)$
$newStateHash \triangleq Hash(sResult.newPublicState, newPrivateStateEnc)$
$newHistoryStateBinding \triangleq Hash(newHistorySummary, newStateHash)$
$newAuthenticator \triangleq GenerateMAC(LL1NVRAM.symmetricKey, newHistoryStateBinding)$
IN

There are two enablement conditions: First, the authenticator supplied by the user (*LL1RAM.authenticator*) must validly bind the user-supplied public and encrypted private state to the history summary supplied by the user. Second, the history summary supplied by the user must match the history summary in the *NVRAM*.

$\land$ $ValidateMAC(LL1NVRAM.symmetricKey, historyStateBinding, LL1RAM.authenticator)$
$\land$ $LL1NVRAM.historySummary = LL1RAM.historySummary$

At the conclusion of the action, the RAM contains the new public and encrypted private state, the new history summary, and an authenticator that binds these together.

$\land$ $LL1RAM' = [$

$publicState \mapsto sResult.newPublicState,$
$privateStateEnc \mapsto newPrivateStateEnc,$
$historySummary \mapsto newHistorySummary,$
$authenticator \mapsto newAuthenticator]$

The $NVRAM$ is updated with the new history summary.

$\land\ LL1NVRAM' = [$
$historySummary \mapsto newHistorySummary,$
$symmetricKey \mapsto LL1NVRAM.symmetricKey]$

The output of the service is added to the set of outputs that the user has observed.

$\land\ LL1ObservedOutputs' = LL1ObservedOutputs \cup \{sResult.output\}$

The disk is unchanged.

$\land$ UNCHANGED $LL1Disk$

The set of available inputs is unchanged

$\land$ UNCHANGED $LL1AvailableInputs$

The new authenticator is added to the set of authenticators that the user has observed.

$\land\ LL1ObservedAuthenticators' =$
$LL1ObservedAuthenticators \cup \{newAuthenticator\}$

The $LL1RepeatOperation$ action is invoked by the user when the computer crashed after a $LL1PerformOperation$ action was performed but before the user had a chance to perform a $LL1WriteDisk$ action to persistently record the new state and its authenticator. This action enables the user to reproduce the result of the most-recent $LL1PerformOperation$ action.

$LL1RepeatOperation \triangleq$
$\exists\, input \in LL1AvailableInputs :$
LET
$stateHash \triangleq Hash(LL1RAM.publicState, LL1RAM.privateStateEnc)$
$historyStateBinding \triangleq Hash(LL1RAM.historySummary, stateHash)$
$privateState \triangleq SymmetricDecrypt(LL1NVRAM.symmetricKey, LL1RAM.privateStateEnc)$
$sResult \triangleq Service(LL1RAM.publicState, privateState, input)$
$newPrivateStateEnc \triangleq$
$SymmetricEncrypt(LL1NVRAM.symmetricKey, sResult.newPrivateState)$
$newStateHash \triangleq Hash(sResult.newPublicState, newPrivateStateEnc)$
$newHistoryStateBinding \triangleq Hash(LL1NVRAM.historySummary, newStateHash)$
$newAuthenticator \triangleq GenerateMAC(LL1NVRAM.symmetricKey, newHistoryStateBinding)$
IN

There are two enablement conditions: First, the authenticator supplied by the user ($LL1RAM.authenticator$) must validly bind the user-supplied public and encrypted private state to the history summary supplied by the user.

Second, the history summary supplied by the user, hashed with the input supplied by the user, must match the history summary in the $NVRAM$. This condition ensures that this action will invoke the service with the same input used with the most recent $LL1PerformOperation$ action.

$\land\ ValidateMAC(LL1NVRAM.symmetricKey, historyStateBinding, LL1RAM.authenticator)$
$\land\ LL1NVRAM.historySummary = Hash(LL1RAM.historySummary, input)$

At the conclusion of the action, the RAM contains the new public and encrypted private state, the new history summary, and an authenticator that binds these together. These should match the values produced by the most recent $LL1PerformOperation$ action.

$\land\ LL1RAM' = [$
$publicState \mapsto sResult.newPublicState,$

$privateStateEnc \mapsto newPrivateStateEnc,$
$historySummary \mapsto LL1NVRAM.historySummary,$
$authenticator \mapsto newAuthenticator]$

The output of the service is added to the set of outputs that the user has observed. If Memoir is working correctly, the user already saw this output when the previous *LL1PerformOperation* action was executed.

$\land\ LL1ObservedOutputs' = LL1ObservedOutputs \cup \{sResult.output\}$

The *NVRAM* is unchanged, because this action is not supposed to change the state of the service.

$\land$ UNCHANGED $LL1NVRAM$

The disk is unchanged.

$\land$ UNCHANGED $LL1Disk$

The set of available inputs is unchanged

$\land$ UNCHANGED $LL1AvailableInputs$

The new authenticator is added to the set of authenticators that the user has observed. If Memoir is working correctly, the user already saw this authenticator when the previous *LL1PerformOperation* action was executed.

$\land\ LL1ObservedAuthenticators' =$
$\quad LL1ObservedAuthenticators \cup \{newAuthenticator\}$

The *LL1Restart* action occurs when the computer restarts.

$LL1Restart \triangleq$
$\exists\ untrustedStorage \in LL1UntrustedStorageType,$
$randomSymmetricKey \in SymmetricKeyType \setminus \{LL1NVRAM.symmetricKey\},$
$hash \in HashType :$

The state of the RAM is *trashed* by a restart, so we set it to some *almost* arbitrary value in *LL1UntrustedStorageType*. The only condition we impose is that the authenticator is not coincidentally equal to an authenticator that could be computed with the symmetric key known only to Memoir.

$\land\ untrustedStorage.authenticator = GenerateMAC(randomSymmetricKey, hash)$
$\land\ LL1RAM' = untrustedStorage$
$\land$ UNCHANGED $LL1Disk$
$\land$ UNCHANGED $LL1NVRAM$
$\land$ UNCHANGED $LL1AvailableInputs$
$\land$ UNCHANGED $LL1ObservedOutputs$
$\land$ UNCHANGED $LL1ObservedAuthenticators$

The *LL1ReadDisk* action copies the state of the disk into the RAM.

$LL1ReadDisk \triangleq$
$\land\ LL1RAM' = LL1Disk$
$\land$ UNCHANGED $LL1Disk$
$\land$ UNCHANGED $LL1NVRAM$
$\land$ UNCHANGED $LL1AvailableInputs$
$\land$ UNCHANGED $LL1ObservedOutputs$
$\land$ UNCHANGED $LL1ObservedAuthenticators$

The *LL1WriteDisk* action copies the state of the RAM onto the disk.

$LL1WriteDisk \triangleq$
$\land$ $LL1Disk' = LL1RAM$
$\land$ UNCHANGED $LL1RAM$
$\land$ UNCHANGED $LL1NVRAM$
$\land$ UNCHANGED $LL1AvailableInputs$
$\land$ UNCHANGED $LL1ObservedOutputs$
$\land$ UNCHANGED $LL1ObservedAuthenticators$

The $LL1CorruptRAM$ action models the ability of a malicious user to attack Memoir by supplying *almost* arbitrary data to Memoir. The data is not completely arbitrary, because the user is assumed to be unable to forge authenticators using the symmetric key stored in the $NVRAM$ of Memoir.

$LL1CorruptRAM \triangleq$
$\exists$ $untrustedStorage \in LL1UntrustedStorageType,$
$fakeSymmetricKey \in SymmetricKeyType \setminus \{LL1NVRAM.symmetricKey\},$
$hash \in HashType$:

The user can launch a replay attack by re-using any authenticator previously observed.

$\land$ $\lor$ $untrustedStorage.authenticator \in LL1ObservedAuthenticators$

Or the user can create a fake authenticator using some other symmetric key.

$\lor$ $untrustedStorage.authenticator = GenerateMAC(fakeSymmetricKey, hash)$
$\land$ $LL1RAM' = untrustedStorage$
$\land$ UNCHANGED $LL1Disk$
$\land$ UNCHANGED $LL1NVRAM$
$\land$ UNCHANGED $LL1AvailableInputs$
$\land$ UNCHANGED $LL1ObservedOutputs$
$\land$ UNCHANGED $LL1ObservedAuthenticators$

The $LL1RestrictedCorruption$ does not model any realistic action in a direct implementation of this low-level spec. The TPM prevents any code other than Memoir from writing to the $NVRAM$, so an attacker cannot actually perform this action in Memoir-Basic.

However, Memoir-Opt allows an attacker to perform an action ($LL2CorruptSPCR$) that corrupts the stored history summary in Memoir-Opt, and in the correctness proof of Memoir-Opt, we will show that the $CorruptSPCR$ action in Memoir-Opt (under some circumstances) refines to the $LL1RestrictedCorruption$ action in Memoir-Basic.

Similarly, when a $LL2Restart$ action occurs in Memoir-Opt, there are circumstances that cause this to refine to the $LL1RestrictedCorruption$ action in Memoir-Basic.

For this reason, we need the $LL1RestrictedCorruption$ action to be strong enough to enable refinement from the $LL2CorruptSPCR$ and $LL2Restart$ actions in Memoir-Opt, but weak enough to enable refinement to an action in the high-level spec, specifically the $HLDie$ action.

We therefore impose a somewhat bizarre-looking pair of constraints on the garbage value to which an attacker can set the history summary in the $NVRAM$. The first constraint (labeled $current$) is needed to ensure that the $CardinalityInvariant$ and $UniquenessInvariant$ continue to hold when a $LL1RestrictedCorruption$ action occurs, and the second constraint (labeled $previous$) is needed to ensure that the $InclusionInvariant$ continues to hold when a $LL1RestrictedCorruption$ action occurs.

$LL1RestrictedCorruption \triangleq$
$\land$ $nvram$::
$\exists\, garbageHistorySummary \in HashType$:
$\land$ $current(garbageHistorySummary)$::

There is no authenticator that validates a history state binding that binds the the garbage history summary to any state hash.

$\forall$ $stateHash \in HashType,\ authenticator \in LL1ObservedAuthenticators$:

LET

$historyStateBinding \triangleq Hash(garbageHistorySummary,\ stateHash)$

IN

$\neg ValidateMAC(LL1NVRAM.symmetricKey,\ historyStateBinding,\ authenticator)$

$\wedge$ $previous(garbageHistorySummary)$::

There is no authenticator that validates a history state binding that binds any predecessor of the the garbage history summary to any state hash.

$\forall$ $stateHash \in HashType,$
$authenticator \in LL1ObservedAuthenticators,$
$someHistorySummary \in HashType,$
$someInput \in InputType$:

LET

$historyStateBinding \triangleq Hash(someHistorySummary,\ stateHash)$

IN

$garbageHistorySummary = Hash(someHistorySummary,\ someInput) \Rightarrow$
$\neg ValidateMAC(LL1NVRAM.symmetricKey,\ historyStateBinding,\ authenticator)$

The history summary in the $NVRAM$ becomes equal to the garbage history summary.

$\wedge$ $LL1NVRAM' = [$
$historySummary \mapsto garbageHistorySummary,$
$symmetricKey \mapsto LL1NVRAM.symmetricKey]$

$\wedge$ ram::

$\vee$ $unchanged$::

A $LL2CorruptSPCR$ action in the Memoir-Opt spec leaves the state of the RAM unchanged.

UNCHANGED $LL1RAM$

$\vee$ $trashed$::

A $LL2Restart$ action in the Memoir-Opt spec trashes the RAM, so we set it to some *almost* arbitrary value in $LL1UntrustedStorageType$. The only condition we impose is that the authenticator is not coincidentally equal to an authenticator that could be computed with the symmetric key known only to Memoir.

$\exists$ $untrustedStorage \in LL1UntrustedStorageType,$
$randomSymmetricKey \in SymmetricKeyType \setminus \{LL1NVRAM.symmetricKey\},$
$hash \in HashType$:

$\wedge$ $untrustedStorage.authenticator = GenerateMAC(randomSymmetricKey,\ hash)$

$\wedge$ $LL1RAM' = untrustedStorage$

$\wedge$ UNCHANGED $LL1Disk$

$\wedge$ UNCHANGED $LL1AvailableInputs$

$\wedge$ UNCHANGED $LL1ObservedOutputs$

$\wedge$ UNCHANGED $LL1ObservedAuthenticators$

In the initial state of the Memoir-Basic implementation, some symmetric key is generated and stored in the $NVRAM$. The initial history summary is the base hash value, indicating that no inputs have been supplied yet. An initial authenticator binds the initial history summary to the initial public and encrypted private state.

$LL1Init \triangleq$

$\exists\, symmetricKey \in SymmetricKeyType$:

LET

$initialPrivateStateEnc \triangleq SymmetricEncrypt(symmetricKey,\ InitialPrivateState)$
$initialStateHash \triangleq Hash(InitialPublicState,\ initialPrivateStateEnc)$

$initialHistoryStateBinding \triangleq Hash(BaseHashValue, initialStateHash)$
$initialAuthenticator \triangleq GenerateMAC(symmetricKey, initialHistoryStateBinding)$
$initialUntrustedStorage \triangleq [$
$publicState \mapsto InitialPublicState,$
$privateStateEnc \mapsto initialPrivateStateEnc,$
$historySummary \mapsto BaseHashValue,$
$authenticator \mapsto initialAuthenticator]$
$initialTrustedStorage \triangleq [$
$historySummary \mapsto BaseHashValue,$
$symmetricKey \mapsto symmetricKey]$
IN
$\land\ LL1Disk = initialUntrustedStorage$
$\land\ LL1RAM = initialUntrustedStorage$
$\land\ LL1NVRAM = initialTrustedStorage$
$\land\ LL1AvailableInputs = InitialAvailableInputs$
$\land\ LL1ObservedOutputs = \{\}$
$\land\ LL1ObservedAuthenticators = \{initialAuthenticator\}$

$LL1Next \triangleq$
$\lor\ LL1MakeInputAvailable$
$\lor\ LL1PerformOperation$
$\lor\ LL1RepeatOperation$
$\lor\ LL1Restart$
$\lor\ LL1ReadDisk$
$\lor\ LL1WriteDisk$
$\lor\ LL1CorruptRAM$
$\lor\ LL1RestrictedCorruption$

$LL1Spec \triangleq LL1Init \land \Box[LL1Next]_{LL1Vars}$

2.5 Specification of the Memoir-Opt System

MODULE *MemoirLL2Specification*

This module defines the specification of Memoir-Opt.

There are nine actions:
LL2MakeInputAvailable
LL2PerformOperation
LL2RepeatOperation
LL2TakeCheckpoint
LL2Restart
LL2ReadDisk
LL2WriteDisk
LL2CorruptRAM
LL2CorruptSPCR

EXTENDS *MemoirLL1Implementation*

In Memoir-Opt, each history summary (which is a composite chained hash of all inputs that the service has processed) is partitioned into two pieces: an anchor and an extension.

HistorySummaryType $\triangleq$ [
anchor : *HashType*,
extension : *HashType*]

The disk and RAM are untrusted. They each can store the service's public state, encrypted private state, a history summary, and an authenticator (a *MAC* that binds the history summary to the public and private state).

LL2UntrustedStorageType $\triangleq$ [
publicState : *PublicStateType*,
privateStateEnc : *PrivateStateEncType*,
historySummary : *HistorySummaryType*,
authenticator : *MACType*]

The *NVRAM* is trusted, since the *TPM* guarantees that it can only be read or written by the code that implements Memoir. The *NVRAM* stores the current history summary anchor and the symmetric key that is used to encrypt the private state and to *MAC* the history summary and service state into an authenticator. It also stores a hash barrier for securing the history summary and a guard bit that indicates whether the current history summary has been extended, such that the history summary anchor is not a complete representation of the inputs summary.

LL2TrustedStorageType $\triangleq$ [
historySummaryAnchor : *HashType*,
symmetricKey : *SymmetricKeyType*,
hashBarrier : *HashType*,
extensionInProgress : BOOLEAN]

LL2AvailableInputs and *LL2ObservedOutputs* are abstract variables that do not directly represent part of the implementation. They correspond to the *HLAvailableInputs* and *HLObservedOutputs* variables in the Memoir-Opt spec.

VARIABLE *LL2AvailableInputs*
VARIABLE *LL2ObservedOutputs*

The *ObservedAuthenticators* variable is also abstract. It records the set of authenticators that the user has seen from Memoir. A malicious user can attempt to use these authenticators in a replay attack against Memoir.

VARIABLE *LL2ObservedAuthenticators*

The $LL2Disk$, $LL2RAM$, $LL2NVRAM$, and $LL2SPCR$ variables represent concrete state maintained by the Memoir-Opt implementation.

The $LL2SPCR$ is semi-trusted. Any party can write to it, but arbitrary writes are not allowed. The only allowable updates are of the form
$\exists\, x : LL2SPCR' = Hash(LL2SPCR, x)$
We use the $LL2SPCR$ to store a history summary extension.

VARIABLE $LL2Disk$
VARIABLE $LL2RAM$
VARIABLE $LL2NVRAM$
VARIABLE $LL2SPCR$

$$
\begin{aligned}
&LL2TypeInvariant \triangleq \\
&\quad \land\ LL2AvailableInputs \subseteq InputType \\
&\quad \land\ LL2ObservedOutputs \subseteq OutputType \\
&\quad \land\ LL2ObservedAuthenticators \subseteq MACType \\
&\quad \land\ LL2Disk \in LL2UntrustedStorageType \\
&\quad \land\ LL2RAM \in LL2UntrustedStorageType \\
&\quad \land\ LL2NVRAM \in LL2TrustedStorageType \\
&\quad \land\ LL2SPCR \in HashType
\end{aligned}
$$

$$
\begin{aligned}
&LL2Vars \triangleq \langle \\
&\quad LL2AvailableInputs, \\
&\quad LL2ObservedOutputs, \\
&\quad LL2ObservedAuthenticators, \\
&\quad LL2Disk, \\
&\quad LL2RAM, \\
&\quad LL2NVRAM, \\
&\quad LL2SPCR\rangle
\end{aligned}
$$

The $Checkpoint$ function takes a history summary that may or may not be checkpointed and produces a checkpointed history summary from it.

$$
\begin{aligned}
&Checkpoint(historySummary) \triangleq \\
&\quad \text{LET} \\
&\qquad checkpointedAnchor \triangleq Hash(historySummary.anchor, historySummary.extension) \\
&\qquad checkpointedHistorySummary \triangleq [\\
&\qquad\quad anchor \mapsto checkpointedAnchor, \\
&\qquad\quad extension \mapsto BaseHashValue] \\
&\quad \text{IN} \\
&\qquad \text{IF } historySummary.extension = BaseHashValue \\
&\qquad\ \ \text{THEN} \\
&\qquad\quad historySummary \\
&\qquad\ \ \text{ELSE} \\
&\qquad\quad checkpointedHistorySummary
\end{aligned}
$$

The $Successor$ function defines the history summary that results from extending a given history summary with a given input. It secures the input using a hash barrier to thwart forgery.

$$
\begin{aligned}
&Successor(historySummary, input, hashBarrier) \triangleq \\
&\quad \text{LET}
\end{aligned}
$$

$securedInput \triangleq Hash(hashBarrier, input)$
$newAnchor \triangleq historySummary.anchor$
$newExtension \triangleq Hash(historySummary.extension, securedInput)$
$newHistorySummary \triangleq [$
$anchor \mapsto newAnchor,$
$extension \mapsto newExtension]$

IN

$newHistorySummary$

The *LL2MakeInputAvailable* action is a direct analog of the *HLMakeInputAvailable* action in the high-level spec.

$LL2MakeInputAvailable \triangleq$
$\exists\, input \in InputType :$
$\land\ input \notin LL2AvailableInputs$
$\land\ LL2AvailableInputs' = LL2AvailableInputs \cup \{input\}$
$\land$ UNCHANGED $LL2Disk$
$\land$ UNCHANGED $LL2RAM$
$\land$ UNCHANGED $LL2NVRAM$
$\land$ UNCHANGED $LL2SPCR$
$\land$ UNCHANGED $LL2ObservedOutputs$
$\land$ UNCHANGED $LL2ObservedAuthenticators$

The *LL2PerformOperation* action is invoked by the user to perform a service operation. It is intended to provide the semantics of the *HLAdvanceService* action in the high-level spec.

$LL2PerformOperation \triangleq$
$\exists\, input \in LL2AvailableInputs :$

LET

$historySummaryHash \triangleq$
$Hash(LL2RAM.historySummary.anchor, LL2RAM.historySummary.extension)$
$stateHash \triangleq Hash(LL2RAM.publicState, LL2RAM.privateStateEnc)$
$historyStateBinding \triangleq Hash(historySummaryHash, stateHash)$
$privateState \triangleq SymmetricDecrypt(LL2NVRAM.symmetricKey, LL2RAM.privateStateEnc)$
$sResult \triangleq Service(LL2RAM.publicState, privateState, input)$
$newPrivateStateEnc \triangleq$
$SymmetricEncrypt(LL2NVRAM.symmetricKey, sResult.newPrivateState)$
$currentHistorySummary \triangleq [$
$anchor \mapsto LL2NVRAM.historySummaryAnchor,$
$extension \mapsto LL2SPCR]$
$newHistorySummary \triangleq Successor(currentHistorySummary, input, LL2NVRAM.hashBarrier)$
$newHistorySummaryHash \triangleq Hash(newHistorySummary.anchor, newHistorySummary.extension)$
$newStateHash \triangleq Hash(sResult.newPublicState, newPrivateStateEnc)$
$newHistoryStateBinding \triangleq Hash(newHistorySummaryHash, newStateHash)$
$newAuthenticator \triangleq GenerateMAC(LL2NVRAM.symmetricKey, newHistoryStateBinding)$

IN

There are three enablement conditions:

First, the authenticator supplied by the user (*LL2RAM.authenticator*) must validly bind the user-supplied public and encrypted private state to the user-supplied history summary.

Second, the value in the $SPCR$ must be consistent with the flag that indicates whether an extension is in progress.

Third, the user-supplied history summary ($LL2RAM.historySummary$) must match the history summary in the TPM ($NVRAM$ and $SPCR$). There are two different ways to check this condition, depending on whether a extension is in progress.

$\land$ $ValidateMAC(LL2NVRAM.symmetricKey,\ historyStateBinding,\ LL2RAM.authenticator)$
$\land$ IF $LL2NVRAM.extensionInProgress = \text{TRUE}$
THEN
$\land$ $LL2SPCR \neq BaseHashValue$
$\land$ $currentHistorySummary = LL2RAM.historySummary$
ELSE
$\land$ $LL2SPCR = BaseHashValue$
$\land$ $currentHistorySummary = Checkpoint(LL2RAM.historySummary)$

At the conclusion of the action, the RAM contains the new public and encrypted private state, the new history summary, and an authenticator that binds these together.

$\land$ $LL2RAM' = [$
$publicState \mapsto sResult.newPublicState,$
$privateStateEnc \mapsto newPrivateStateEnc,$
$historySummary \mapsto newHistorySummary,$
$authenticator \mapsto newAuthenticator]$

The $NVRAM$ is updated to indicate that the extension is in progress. The $NVRAM$ may already indicate this, in which case this step does not require writing to the $NVRAM$.

$\land$ $LL2NVRAM' = [$
$historySummaryAnchor \mapsto LL2NVRAM.historySummaryAnchor,$
$symmetricKey \mapsto LL2NVRAM.symmetricKey,$
$hashBarrier \mapsto LL2NVRAM.hashBarrier,$
$extensionInProgress \mapsto \text{TRUE}]$

The $SPCR$ is updated with the new history summary extension.

$\land$ $LL2SPCR' = newHistorySummary.extension$

The output of the service is added to the set of outputs that the user has observed.

$\land$ $LL2ObservedOutputs' = LL2ObservedOutputs \cup \{sResult.output\}$

The disk is unchanged.

$\land$ UNCHANGED $LL2Disk$

The set of available inputs is unchanged.

$\land$ UNCHANGED $LL2AvailableInputs$

The new authenticator is added to the set of authenticators that the user has observed.

$\land$ $LL2ObservedAuthenticators' =$
$LL2ObservedAuthenticators \cup \{newAuthenticator\}$

The *LL2RepeatOperation* action is invoked by the user when the computer crashed after a *LL2PerformOperation* action was performed but before the user had a chance to perform a *LL2WriteDisk* action to persistently record the new state and its authenticator. This action enables the user to reproduce the result of the most-recent *LL2PerformOperation* action.

$LL2RepeatOperation \triangleq$
$\exists\, input \in LL2AvailableInputs :$
LET
$historySummaryHash \triangleq$
$Hash(LL2RAM.historySummary.anchor,\ LL2RAM.historySummary.extension)$

$stateHash \triangleq Hash(LL2RAM.publicState, LL2RAM.privateStateEnc)$
$historyStateBinding \triangleq Hash(historySummaryHash, stateHash)$
$newHistorySummary \triangleq Successor(LL2RAM.historySummary, input, LL2NVRAM.hashBarrier)$
$checkpointedHistorySummary \triangleq Checkpoint(LL2RAM.historySummary)$
$newCheckpointedHistorySummary \triangleq$
$\quad Successor(checkpointedHistorySummary, input, LL2NVRAM.hashBarrier)$
$checkpointedNewHistorySummary \triangleq Checkpoint(newHistorySummary)$
$checkpointedNewCheckpointedHistorySummary \triangleq$
$\quad Checkpoint(newCheckpointedHistorySummary)$
$privateState \triangleq SymmetricDecrypt(LL2NVRAM.symmetricKey, LL2RAM.privateStateEnc)$
$sResult \triangleq Service(LL2RAM.publicState, privateState, input)$
$newPrivateStateEnc \triangleq$
$\quad SymmetricEncrypt(LL2NVRAM.symmetricKey, sResult.newPrivateState)$
$currentHistorySummary \triangleq [$
$\quad anchor \mapsto LL2NVRAM.historySummaryAnchor,$
$\quad extension \mapsto LL2SPCR]$
$currentHistorySummaryHash \triangleq Hash(LL2NVRAM.historySummaryAnchor, LL2SPCR)$
$newStateHash \triangleq Hash(sResult.newPublicState, newPrivateStateEnc)$
$newHistoryStateBinding \triangleq Hash(currentHistorySummaryHash, newStateHash)$
$newAuthenticator \triangleq GenerateMAC(LL2NVRAM.symmetricKey, newHistoryStateBinding)$

IN

There are three enablement conditions:

First, the authenticator supplied by the user (*LL2RAM.authenticator*) must validly bind the user-supplied public and encrypted private state to the history summary supplied by the user.

Second, a *TakeCheckpoint* action should always occur immediately before a shutdown, power-off, or reboot; therefore, an extension will not be in progress at the time *LL2RepeatOperation* is needed. This implies that the flag in the *NVRAM* must indicate that an extension is not in progress, and the value in the *SPCR* must equal the *BaseHashValue*.

Third, the user-supplied history summary (*LL2RAM.historySummary*), extended with the user-supplied input, must match the history summary in the *TPM* (*NVRAM* and *SPCR*). This condition ensures that this action will invoke the service with the same input used with the most recent *LL2PerformOperation* action. There are two different ways to check this condition, depending on whether a checkpoint was taken before the input was processed.

$\land\ ValidateMAC(LL2NVRAM.symmetricKey, historyStateBinding, LL2RAM.authenticator)$
$\land\ LL2NVRAM.extensionInProgress = \text{FALSE}$
$\land\ LL2SPCR = BaseHashValue$
$\land$ no checkpoint before input
$\quad \lor\ currentHistorySummary = checkpointedNewHistorySummary$
$\quad$ checkpoints before input
$\quad \lor\ currentHistorySummary = checkpointedNewCheckpointedHistorySummary$

At the conclusion of the action, the RAM contains the new public and encrypted private state, the new history summary, and an authenticator that binds these together. These should match the values produced by the most recent *LL2PerformOperation* action.

$\land\ LL2RAM' = [$
$\quad publicState \mapsto sResult.newPublicState,$
$\quad privateStateEnc \mapsto newPrivateStateEnc,$
$\quad historySummary \mapsto currentHistorySummary,$
$\quad authenticator \mapsto newAuthenticator]$

The output of the service is added to the set of outputs that the user has observed. If Memoir is working correctly, the user already saw this output when the previous *LL2PerformOperation* action was executed.

$\land\ LL2ObservedOutputs' = LL2ObservedOutputs \cup \{sResult.output\}$

The $NVRAM$ is unchanged, because this action is not supposed to change the state of the service.

$\land$ UNCHANGED $LL2NVRAM$

The $SPCR$ is unchanged, because this action is not supposed to change the state of the service.

$\land$ UNCHANGED $LL2SPCR$

The disk is unchanged.

$\land$ UNCHANGED $LL2Disk$

The set of available inputs is unchanged.

$\land$ UNCHANGED $LL2AvailableInputs$

The new authenticator is added to the set of authenticators that the user has observed. If Memoir is working correctly, the user already saw this authenticator when the previous $LL2PerformOperation$ action was executed.

$\land$ $LL2ObservedAuthenticators' =$
$\quad LL2ObservedAuthenticators \cup \{newAuthenticator\}$

The $LL2TakeCheckpoint$ action occurs in response to an NMI indicating that a shutdown, power-off, or reboot is imminent.

$LL2TakeCheckpoint \triangleq$

LET

$\quad newHistorySummaryAnchor \triangleq Hash(LL2NVRAM.historySummaryAnchor, LL2SPCR)$

IN

There are two enablement conditions. The guard bit in the $NVRAM$ must indicate that an extension is in progress, and the $SPCR$ must contain an extension.

$\land$ $LL2NVRAM.extensionInProgress = \text{TRUE}$
$\land$ $LL2SPCR \neq BaseHashValue$

This action changes nothing other than the $NVRAM$.

$\land$ UNCHANGED $LL2RAM$
$\land$ UNCHANGED $LL2Disk$
$\land$ $LL2NVRAM' = [$
$\quad historySummaryAnchor \mapsto newHistorySummaryAnchor,$
$\quad symmetricKey \mapsto LL2NVRAM.symmetricKey,$
$\quad hashBarrier \mapsto LL2NVRAM.hashBarrier,$
$\quad extensionInProgress \mapsto \text{FALSE}]$
$\land$ UNCHANGED $LL2SPCR$
$\land$ UNCHANGED $LL2AvailableInputs$
$\land$ UNCHANGED $LL2ObservedOutputs$
$\land$ UNCHANGED $LL2ObservedAuthenticators$

The $LL2Restart$ action occurs when the computer restarts.

$LL2Restart \triangleq$

$\exists$ $untrustedStorage \in LL2UntrustedStorageType,$
$\quad randomSymmetricKey \in SymmetricKeyType \setminus \{LL2NVRAM.symmetricKey\},$
$\quad hash \in HashType :$

The state of the RAM is garbaged by a restart, so we set it to some *almost* arbitrary value in $LL2UntrustedStorageType$. The only condition we impose is that the authenticator is not coincidentally equal to an authenticator that could be computed with the symmetric key known only to Memoir.

$\land$ $untrustedStorage.authenticator = GenerateMAC(randomSymmetricKey, hash)$
$\land$ $LL2RAM' = untrustedStorage$

$\wedge$ UNCHANGED $LL2Disk$
$\wedge$ UNCHANGED $LL2NVRAM$

The value of the $SPCR$ is set to a known starting value, which we model with the $BaseHashValue$.

$\wedge$ $LL2SPCR' = BaseHashValue$
$\wedge$ UNCHANGED $LL2AvailableInputs$
$\wedge$ UNCHANGED $LL2ObservedOutputs$
$\wedge$ UNCHANGED $LL2ObservedAuthenticators$

The $LL2ReadDisk$ action copies the state of the disk into the RAM.

$LL2ReadDisk \triangleq$
$\wedge$ $LL2RAM' = LL2Disk$
$\wedge$ UNCHANGED $LL2Disk$
$\wedge$ UNCHANGED $LL2NVRAM$
$\wedge$ UNCHANGED $LL2SPCR$
$\wedge$ UNCHANGED $LL2AvailableInputs$
$\wedge$ UNCHANGED $LL2ObservedOutputs$
$\wedge$ UNCHANGED $LL2ObservedAuthenticators$

The $LL2WriteDisk$ action copies the state of the RAM onto the disk.

$LL2WriteDisk \triangleq$
$\wedge$ $LL2Disk' = LL2RAM$
$\wedge$ UNCHANGED $LL2RAM$
$\wedge$ UNCHANGED $LL2NVRAM$
$\wedge$ UNCHANGED $LL2SPCR$
$\wedge$ UNCHANGED $LL2AvailableInputs$
$\wedge$ UNCHANGED $LL2ObservedOutputs$
$\wedge$ UNCHANGED $LL2ObservedAuthenticators$

The $LL2CorruptRAM$ action models the ability of a malicious user to attack Memoir by supplying *almost* arbitrary data to Memoir. The data is not completely arbitrary, because the user is assumed to be unable to forge authenticators using the symmetric key stored in the $NVRAM$ of the TPM.

$LL2CorruptRAM \triangleq$
$\exists$ $untrustedStorage \in LL2UntrustedStorageType$,
$fakeSymmetricKey \in SymmetricKeyType \setminus \{LL2NVRAM.symmetricKey\}$,
$hash \in HashType$:

The user can launch a replay attack by re-using any authenticator previously observed.

$\wedge$ $\vee$ $untrustedStorage.authenticator \in LL2ObservedAuthenticators$

Or the user can create a fake authenticator using some other symmetric key.

$\vee$ $untrustedStorage.authenticator = GenerateMAC(fakeSymmetricKey, hash)$
$\wedge$ $LL2RAM' = untrustedStorage$
$\wedge$ UNCHANGED $LL2Disk$
$\wedge$ UNCHANGED $LL2NVRAM$
$\wedge$ UNCHANGED $LL2SPCR$
$\wedge$ UNCHANGED $LL2AvailableInputs$
$\wedge$ UNCHANGED $LL2ObservedOutputs$

$\wedge$ UNCHANGED $LL2ObservedAuthenticators$

The *LL2CorruptSPCR* action models the ability of a malicious user to attack Memoir by extending the *SPCR* with *almost* arbitrary data. The data is not completely arbitrary, because the user is assumed not to know the hash barrier stored in the *NVRAM* of the *TPM*, so the user has negligible probability of being able to correctly guess this value and use it to extend the *SPCR*.

$LL2CorruptSPCR \triangleq$
$\quad \exists\, fakeHash \in HashDomain :$
$\quad\quad$ LET
$\quad\quad\quad$ The *SPCR* can only be modified by extending it.
$\quad\quad\quad newHistorySummaryExtension \triangleq Hash(LL2SPCR, fakeHash)$
$\quad\quad$ IN
$\quad\quad\quad \wedge\ \forall\, fakeInput \in InputType : fakeHash \neq Hash(LL2NVRAM.hashBarrier, fakeInput)$
$\quad\quad\quad \wedge$ UNCHANGED $LL2RAM$
$\quad\quad\quad \wedge$ UNCHANGED $LL2Disk$
$\quad\quad\quad \wedge$ UNCHANGED $LL2NVRAM$
$\quad\quad\quad \wedge\ LL2SPCR' = newHistorySummaryExtension$
$\quad\quad\quad \wedge$ UNCHANGED $LL2AvailableInputs$
$\quad\quad\quad \wedge$ UNCHANGED $LL2ObservedOutputs$
$\quad\quad\quad \wedge$ UNCHANGED $LL2ObservedAuthenticators$

In the initial state of the Memoir-Opt spec, some symmetric key and some hash barrier are generated and stored in the *NVRAM*. The initial history summary anchor and extension are both the base hash value, indicating that no inputs have been supplied yet. An initial authenticator binds the initial history summary to the initial public and encrypted private state.

$LL2Init \triangleq$
$\quad \exists\, symmetricKey \in SymmetricKeyType,\ hashBarrier \in HashType :$
$\quad\quad$ LET
$\quad\quad\quad initialPrivateStateEnc \triangleq SymmetricEncrypt(symmetricKey, InitialPrivateState)$
$\quad\quad\quad initialStateHash \triangleq Hash(InitialPublicState, initialPrivateStateEnc)$
$\quad\quad\quad initialHistorySummary \triangleq [$
$\quad\quad\quad\quad anchor \mapsto BaseHashValue,$
$\quad\quad\quad\quad extension \mapsto BaseHashValue]$
$\quad\quad\quad initialHistorySummaryHash \triangleq Hash(BaseHashValue, BaseHashValue)$
$\quad\quad\quad initialHistoryStateBinding \triangleq Hash(initialHistorySummaryHash, initialStateHash)$
$\quad\quad\quad initialAuthenticator \triangleq GenerateMAC(symmetricKey, initialHistoryStateBinding)$
$\quad\quad\quad initialUntrustedStorage \triangleq [$
$\quad\quad\quad\quad publicState \mapsto InitialPublicState,$
$\quad\quad\quad\quad privateStateEnc \mapsto initialPrivateStateEnc,$
$\quad\quad\quad\quad historySummary \mapsto initialHistorySummary,$
$\quad\quad\quad\quad authenticator \mapsto initialAuthenticator]$
$\quad\quad\quad initialTrustedStorage \triangleq [$
$\quad\quad\quad\quad historySummaryAnchor \mapsto BaseHashValue,$
$\quad\quad\quad\quad symmetricKey \mapsto symmetricKey,$
$\quad\quad\quad\quad hashBarrier \mapsto hashBarrier,$
$\quad\quad\quad\quad extensionInProgress \mapsto$ FALSE$]$
$\quad\quad$ IN
$\quad\quad\quad \wedge\ LL2Disk = initialUntrustedStorage$

$\land$ $LL2RAM = initialUntrustedStorage$
$\land$ $LL2NVRAM = initialTrustedStorage$
$\land$ $LL2SPCR = BaseHashValue$
$\land$ $LL2AvailableInputs = InitialAvailableInputs$
$\land$ $LL2ObservedOutputs = \{\}$
$\land$ $LL2ObservedAuthenticators = \{initialAuthenticator\}$

$LL2Next \triangleq$
$\lor$ $LL2MakeInputAvailable$
$\lor$ $LL2PerformOperation$
$\lor$ $LL2RepeatOperation$
$\lor$ $LL2TakeCheckpoint$
$\lor$ $LL2Restart$
$\lor$ $LL2ReadDisk$
$\lor$ $LL2WriteDisk$
$\lor$ $LL2CorruptRAM$
$\lor$ $LL2CorruptSPCR$

$LL2Spec \triangleq LL2Init \land \Box[LL2Next]_{LL2Vars}$

3. REFINEMENTS AND INVARIANTS

This section presents two classes of TLA+ modules. First, it contains modules pertaining to the refinement of one spec to another. There are two such modules, one that refines the Memoir-Basic low-level spec to the high-level spec and one that refines the Memoir-Opt low-level spec to the Memoir-Basic low-level spec.

Second, this section includes modules describing invariants maintained by the Memoir-Basic spec. This include both invariants needed for proving the Memoir-Basic refinement and also invariants needed for proving those invariants. No invariants are necessary for proving Memoir-Opt refinement.

3.1 Refinement 1: Mapping Memoir-Basic State to High-Level State

MODULE *MemoirLL1Refinement*

This module describes how to interpret the state of the Memoir-Basic spec as a state of the high-level spec.

This module includes the following definitions:
LL1HistoryStateBindingAuthenticated
LL1NVRAMHistorySummaryUncorrupted
LL1Refinement

EXTENDS *MemoirLL1Specification*

The *LL1HistoryStateBindingAuthenticated* predicate asserts that a history state binding is authenticated, meaning that the set of observed authenticators includes an authenticator that is a valid *MAC* of this history state binding.

$$
\begin{array}{l}
LL1HistoryStateBindingAuthenticated(historyStateBinding) \triangleq \\
\quad \exists\, authenticator \in LL1ObservedAuthenticators : \\
\qquad ValidateMAC(LL1NVRAM.symmetricKey,\ historyStateBinding,\ authenticator)
\end{array}
$$

The *LL1NVRAMHistorySummaryUncorrupted* predicate asserts that there exists some state hash that is bound to the history summary in the *NVRAM* by an authenticated history state binding. This predicate is initially true, and it remains true until a *LL1RestrictedCorruption* action occurs, which makes the predicate false, and it remains false thereafter.

$$
\begin{array}{l}
LL1NVRAMHistorySummaryUncorrupted \triangleq \\
\quad \exists\, stateHash \in HashType : \\
\qquad \text{LET} \\
\qquad\quad historyStateBinding \triangleq Hash(LL1NVRAM.historySummary,\ stateHash) \\
\qquad \text{IN} \\
\qquad\quad LL1HistoryStateBindingAuthenticated(historyStateBinding)
\end{array}
$$

The *LL1Refinement* describes the relationship between the Memoir-Basic spec and the high-level spec.

$LL1Refinement \triangleq$

The high-level available inputs correspond exactly to the Memoir-Basic available inputs, since both are abstractions that model the availability of particular input values to the user.

$\land\ HLAvailableInputs = LL1AvailableInputs$

The high-level observed outputs correspond exactly to the Memoir-Basic observed outputs, since both are abstractions that model the set of outputs that the user has so far observed from the operation of the service.

$\land\ HLObservedOutputs = LL1ObservedOutputs$

The high-level public and private state is defined in terms of the history summary in the *NVRAM*. There are two possibilities.

$\land$ IF $LL1NVRAMHistorySummaryUncorrupted$

THEN

First, if the set of observed authenticators contains an authenticator that binds any state hash to the history summary currently in the *NVRAM*, then the public and private state is any state of the legal type whose hash is so bound. In this case, the service is alive.

$$
\begin{array}{l}
\text{LET} \\
\quad refPrivateStateEnc \triangleq SymmetricEncrypt(LL1NVRAM.symmetricKey,\ HLPrivateState) \\
\quad refStateHash \triangleq Hash(HLPublicState,\ refPrivateStateEnc) \\
\quad refHistoryStateBinding \triangleq Hash(LL1NVRAM.historySummary,\ refStateHash) \\
\text{IN} \\
\quad \land\ HLPublicState \in PublicStateType \\
\quad \land\ HLPrivateState \in PrivateStateType \\
\quad \land\ LL1HistoryStateBindingAuthenticated(refHistoryStateBinding)
\end{array}
$$

$\wedge \ HLAlive = \text{TRUE}$

ELSE

Second, if the set of observed authenticators does not contain an authenticator that binds any state hash to the history summary currently in the $NVRAM$, then the values of the public and private state are equal to their dead states, and the service is not alive.

$\wedge \ HLPublicState = DeadPublicState$

$\wedge \ HLPrivateState = DeadPrivateState$

$\wedge \ HLAlive = \text{FALSE}$

3.2 Refinement 2: Mapping Memoir-Opt State to Memoir-Basic State

MODULE *MemoirLL2Refinement*

This module describes how to interpret the state of Memoir-Opt as a state of Memoir-Basic.

This module includes the following definitions:
LL2HistorySummaryIsSuccessor
HistorySummariesMatch
AuthenticatorsMatch
AuthenticatorSetsMatch
LL2NVRAMLogicalHistorySummary
LL2Refinement

EXTENDS *MemoirLL2Specification*, *Sequences*

The *LL2HistorySummaryIsSuccessor* predicate defines, in the Memoir-Opt spec, the conditions under which one history summary is a successor of another history summary with a particular intervening input.

$$
\begin{array}{l}
\mathit{LL2HistorySummaryIsSuccessor}(\mathit{historySummary}, \mathit{previousHistorySummary}, \mathit{input}, \mathit{hashBarrier}) \triangleq \\
\quad \text{LET} \\
\qquad \mathit{successorHistorySummary} \triangleq \mathit{Successor}(\mathit{previousHistorySummary}, \mathit{input}, \mathit{hashBarrier}) \\
\qquad \mathit{checkpointedSuccessorHistorySummary} \triangleq \mathit{Checkpoint}(\mathit{successorHistorySummary}) \\
\quad \text{IN} \\
\qquad \lor \mathit{historySummary} = \mathit{successorHistorySummary} \\
\qquad \lor \mathit{historySummary} = \mathit{checkpointedSuccessorHistorySummary}
\end{array}
$$

The *HistorySummariesMatch* predicate defines the conditions under which a history summary in the Memoir-Basic spec semantically matches a history summary in the Memoir-Opt spec.

This requires a recursive definition, but the current version of the prover cannot handle recursive operators, nor can it tractably support proofs using recursive function definitions. Therefore, we define the operator indirectly, by using an assumption. Although *Lamport* has stated that this approach is “not a satisfactory alternative to recursive definitions,” he has also called it “a reasonable hack to get a proof done.”

CONSTANT $\mathit{HistorySummariesMatch}(\_, \_, \_)$

$$
\begin{array}{l}
\mathit{HistorySummariesMatchRecursion}(\mathit{ll1HistorySummary}, \mathit{ll2HistorySummary}, \mathit{hashBarrier}) \triangleq \\
\quad \exists\ \mathit{input} \in \mathit{InputType}, \\
\qquad \mathit{previousLL1HistorySummary} \in \mathit{HashType}, \\
\qquad \mathit{previousLL2HistorySummary} \in \mathit{HistorySummaryType} : \\
\qquad \land\ \mathit{HistorySummariesMatch}(\mathit{previousLL1HistorySummary}, \mathit{previousLL2HistorySummary}, \mathit{hashBarrier}) \\
\qquad \land\ \mathit{ll1HistorySummary} = \mathit{Hash}(\mathit{previousLL1HistorySummary}, \mathit{input}) \\
\qquad \land\ \mathit{LL2HistorySummaryIsSuccessor}(\\
\qquad\qquad \mathit{ll2HistorySummary}, \mathit{previousLL2HistorySummary}, \mathit{input}, \mathit{hashBarrier})
\end{array}
$$

$$
\begin{array}{l}
\text{ASSUME } \mathit{HistorySummariesMatchDefinition} \triangleq \\
\quad \forall\ \mathit{ll1HistorySummary} \in \mathit{HashType}, \\
\qquad \mathit{ll2HistorySummary} \in \mathit{HistorySummaryType}, \\
\qquad \mathit{hashBarrier} \in \mathit{HashType} : \\
\qquad \text{LET} \\
\qquad\quad \mathit{ll2InitialHistorySummary} \triangleq [\mathit{anchor} \mapsto \mathit{BaseHashValue}, \mathit{extension} \mapsto \mathit{BaseHashValue}] \\
\qquad \text{IN} \\
\qquad\quad \text{IF } \mathit{ll2HistorySummary} = \mathit{ll2InitialHistorySummary} \\
\qquad\quad\ \text{THEN} \\
\qquad\qquad \mathit{HistorySummariesMatch}(\mathit{ll1HistorySummary}, \mathit{ll2HistorySummary}, \mathit{hashBarrier}) =
\end{array}
$$

$(\mathit{ll1HistorySummary} = \mathit{BaseHashValue})$

ELSE

$\mathit{HistorySummariesMatch}(\mathit{ll1HistorySummary}, \mathit{ll2HistorySummary}, \mathit{hashBarrier}) =$
$\quad \mathit{HistorySummariesMatchRecursion}(\mathit{ll1HistorySummary}, \mathit{ll2HistorySummary}, \mathit{hashBarrier})$

The *AuthenticatorsMatch* predicate defines the conditions under which an authenticator in the Memoir-Basic spec semantically matches an authenticator in the Memoir-Opt spec.

$\mathit{AuthenticatorsMatch}(\mathit{ll1Authenticator}, \mathit{ll2Authenticator}, \mathit{symmetricKey}, \mathit{hashBarrier}) \triangleq$
$\quad \exists\ \mathit{stateHash} \in \mathit{HashType},$
$\quad\quad \mathit{ll1HistorySummary} \in \mathit{HashType},$
$\quad\quad \mathit{ll2HistorySummary} \in \mathit{HistorySummaryType} :$

LET

$\mathit{ll1HistoryStateBinding} \triangleq \mathit{Hash}(\mathit{ll1HistorySummary}, \mathit{stateHash})$
$\mathit{ll2HistorySummaryHash} \triangleq \mathit{Hash}(\mathit{ll2HistorySummary.anchor}, \mathit{ll2HistorySummary.extension})$
$\mathit{ll2HistoryStateBinding} \triangleq \mathit{Hash}(\mathit{ll2HistorySummaryHash}, \mathit{stateHash})$

IN

The Memoir-Opt authenticator is a valid *MAC* of an Memoir-Opt history state binding that binds some Memoir-Opt history summary to some state hash.

$\wedge\ \mathit{ValidateMAC}(\mathit{symmetricKey}, \mathit{ll2HistoryStateBinding}, \mathit{ll2Authenticator})$

The Memoir-Basic authenticator is generated as a *MAC* of a Memoir-Basic history state binding that binds some Memoir-Basic history summary to the same state hash as the previous conjunct.

$\wedge\ \mathit{ll1Authenticator} = \mathit{GenerateMAC}(\mathit{symmetricKey}, \mathit{ll1HistoryStateBinding})$

The Memoir-Basic history state binding matches the Memoir-Opt history state binding.

$\wedge\ \mathit{HistorySummariesMatch}(\mathit{ll1HistorySummary}, \mathit{ll2HistorySummary}, \mathit{hashBarrier})$

The *AuthenticatorSetsMatch* predicate defines the conditions under which a set of authenticators in the Memoir-Basic spec semantically matches a set of authenticators in the Memoir-Opt spec.

$\mathit{AuthenticatorSetsMatch}(\mathit{ll1Authenticators}, \mathit{ll2Authenticators}, \mathit{symmetricKey}, \mathit{hashBarrier}) \triangleq$

For every authenticator in the Memoir-Basic set, there is some authenticator in the Memoir-Opt set that matches it.

$\wedge\ \forall\, \mathit{ll1Authenticator} \in \mathit{ll1Authenticators} :$
$\quad \exists\, \mathit{ll2Authenticator} \in \mathit{ll2Authenticators} :$
$\quad\quad \mathit{AuthenticatorsMatch}($
$\quad\quad\quad \mathit{ll1Authenticator}, \mathit{ll2Authenticator}, \mathit{symmetricKey}, \mathit{hashBarrier})$

For every authenticator in the Memoir-Opt set, there is some authenticator in the Memoir-Basic set that matches it.

$\wedge\ \forall\, \mathit{ll2Authenticator} \in \mathit{ll2Authenticators} :$
$\quad \exists\, \mathit{ll1Authenticator} \in \mathit{ll1Authenticators} :$
$\quad\quad \mathit{AuthenticatorsMatch}($
$\quad\quad\quad \mathit{ll1Authenticator}, \mathit{ll2Authenticator}, \mathit{symmetricKey}, \mathit{hashBarrier})$

The *LL2NVRAMLogicalHistorySummary* is the history summary that is represented by the state of the *NVRAM* and the *SPCR* in the Memoir-Opt spec. The anchor always comes directly from the *NVRAM*, but the extension only comes from the *SPCR* if the *NVRAM* indicates that an extension is in progress; otherwise, the extension is set to the base hash value. The reason for this is that a *LL2TakeCheckpoint* action clears the *extensionInProgress* flag but is unable to reset the *SPCR*, so during the time between a *LL2TakeCheckpoint* action and a *LL2Restart* action, the extension is really the base hash value, even though the *SPCR* has not yet been reset.

The other interesting aspect of *LL2NVRAMLogicalHistorySummary* is that if there is an extension in progress but the *SPCR* equals the base hash value, then the logical value of the extension is set to a crazy hash value. This is necessary because if a Restart occurs when the *LL2NVRAM.historySummaryAnchor* equals the base hash value, the *SPCR* will be set to the *BaseHashValue*, but we don't want the logical history summary to appear the same as the initial history summary, which it would if both the anchor and the extension were to equal the base hash value.

$\textsc{constant}\ \ CrazyHashValue$

$\textsc{assume}\ \ CrazyHashValueTypeSafe \triangleq CrazyHashValue \in HashType$

$$
\begin{array}{l}
\textsc{assume}\ \ CrazyHashValueUnique \triangleq \\
\quad \land\ \forall\, hashInput1,\ hashInput2 \in HashDomain : \\
\qquad\quad Hash(hashInput1,\ hashInput2) \neq CrazyHashValue \\
\quad \land\ BaseHashValue \neq CrazyHashValue
\end{array}
$$

$$
\begin{array}{l}
LL2NVRAMLogicalHistorySummary \triangleq \\
\quad \textsc{if}\ LL2NVRAM.extensionInProgress \\
\quad\ \textsc{then} \\
\qquad \textsc{if}\ LL2SPCR = BaseHashValue \\
\qquad\ \textsc{then} \\
\qquad\quad [anchor \mapsto LL2NVRAM.historySummaryAnchor, \\
\qquad\quad\ extension \mapsto CrazyHashValue] \\
\qquad\ \textsc{else} \\
\qquad\quad [anchor \mapsto LL2NVRAM.historySummaryAnchor, \\
\qquad\quad\ extension \mapsto LL2SPCR] \\
\quad\ \textsc{else} \\
\quad\quad [anchor \mapsto LL2NVRAM.historySummaryAnchor, \\
\quad\quad\ extension \mapsto BaseHashValue]
\end{array}
$$

The *LL2Refinement* describes the relationship between the $2nd$ low-level spec (Memoir-Opt) and the $1st$ low-level spec (Memoir-Basic).

The following variables are directly equal between the two specs:
LLxAvailableInputs
LLxObservedOutputs
LLxDisk.publicState
LLxDisk.privateStateEnc
LLxRAM.publicState
LLxRAM.privateStateEnc
LLxNVRAM.symmetricKey

The following variables directly match according to the operators defined above:
LLxObservedAuthenticators
LLxDisk.historySummary
LLxDisk.authenticator
LLxRAM.historySummary
LLxRAM.authenticator
Note that the authenticators in the sets of observed authenticators match using the symmetric key in the *LL2NVRAM*. By contrast, the authenticators in the disk and RAM match using some unspecified symmetric key, because the values in the disk and RAM may be set arbitrarily by the user.

The following variable has a more involved matching process, involving both the *LL2NVRAMLogicalHistorySummary* operator and a match operator:

LL1NVRAM.historySummary

For each of the variables that are refined via match predicates rather than through an equality relation, the refinement asserts that the variable has the appropriate type.

For each of the record types (*LL1Disk*, *LL1RAM*, and *LL1NVRAM*), the refinement defines the mapping for each individual field within the record.

$$
\begin{array}{l}
\mathit{LL2Refinement} \triangleq \\
\quad \land\ \mathit{LL1AvailableInputs} = \mathit{LL2AvailableInputs} \\
\quad \land\ \mathit{LL1ObservedOutputs} = \mathit{LL2ObservedOutputs} \\
\quad \land\ \mathit{LL1ObservedAuthenticators} \subseteq \mathit{MACType} \\
\quad \land\ \mathit{AuthenticatorSetsMatch}(\\
\qquad\quad \mathit{LL1ObservedAuthenticators}, \\
\qquad\quad \mathit{LL2ObservedAuthenticators}, \\
\qquad\quad \mathit{LL2NVRAM.symmetricKey}, \\
\qquad\quad \mathit{LL2NVRAM.hashBarrier}) \\
\quad \land\ \mathit{LL1Disk} \in \mathit{LL1UntrustedStorageType} \\
\quad \land\ \mathit{LL1Disk.publicState} = \mathit{LL2Disk.publicState} \\
\quad \land\ \mathit{LL1Disk.privateStateEnc} = \mathit{LL2Disk.privateStateEnc} \\
\quad \land\ \mathit{HistorySummariesMatch}(\\
\qquad\quad \mathit{LL1Disk.historySummary}, \\
\qquad\quad \mathit{LL2Disk.historySummary}, \\
\qquad\quad \mathit{LL2NVRAM.hashBarrier}) \\
\quad \land\ \exists\, \mathit{symmetricKey} \in \mathit{SymmetricKeyType} : \\
\qquad \mathit{AuthenticatorsMatch}(\\
\qquad\qquad \mathit{LL1Disk.authenticator}, \\
\qquad\qquad \mathit{LL2Disk.authenticator}, \\
\qquad\qquad \mathit{symmetricKey}, \\
\qquad\qquad \mathit{LL2NVRAM.hashBarrier}) \\
\quad \land\ \mathit{LL1RAM} \in \mathit{LL1UntrustedStorageType} \\
\quad \land\ \mathit{LL1RAM.publicState} = \mathit{LL2RAM.publicState} \\
\quad \land\ \mathit{LL1RAM.privateStateEnc} = \mathit{LL2RAM.privateStateEnc} \\
\quad \land\ \mathit{HistorySummariesMatch}(\\
\qquad\quad \mathit{LL1RAM.historySummary}, \\
\qquad\quad \mathit{LL2RAM.historySummary}, \\
\qquad\quad \mathit{LL2NVRAM.hashBarrier}) \\
\quad \land\ \exists\, \mathit{symmetricKey} \in \mathit{SymmetricKeyType} : \\
\qquad \mathit{AuthenticatorsMatch}(\\
\qquad\qquad \mathit{LL1RAM.authenticator}, \\
\qquad\qquad \mathit{LL2RAM.authenticator}, \\
\qquad\qquad \mathit{symmetricKey}, \\
\qquad\qquad \mathit{LL2NVRAM.hashBarrier}) \\
\quad \land\ \mathit{LL1NVRAM} \in \mathit{LL1TrustedStorageType} \\
\quad \land\ \mathit{HistorySummariesMatch}(\\
\qquad\quad \mathit{LL1NVRAM.historySummary}, \\
\qquad\quad \mathit{LL2NVRAMLogicalHistorySummary}, \\
\qquad\quad \mathit{LL2NVRAM.hashBarrier}) \\
\quad \land\ \mathit{LL1NVRAM.symmetricKey} = \mathit{LL2NVRAM.symmetricKey}
\end{array}
$$

3.3 Invariants Needed to Prove Memoir-Basic Implementation

MODULE *MemoirLL1CorrectnessInvariants*

This module defines the three correctness invariants needed to prove that the Memoir-Basic spec implements the high-level spec.

EXTENDS *MemoirLL1TypeSafety*

The *UnforgeabilityInvariant* states that, for any authenticator residing in the user's RAM, if the authenticator validates using the symmetric key in the *NVRAM*, the authenticator is in the set of authenticators that the user observed Memoir to produce.

This is a somewhat boring invariant. It really just extends the assumption in the *LL1CorruptRAM* action, which constraints the set of authenticators the user can create. If we had written the low-level spec differently, such that this constraint had been expressed in the *LL1PerformOperation* and *LL1RepeatOperation* actions instead of the *LL1CorruptRAM* action, this invariant might not be necessary.

$$
\begin{array}{l}
\mathit{UnforgeabilityInvariant} \triangleq \\
\quad \forall\, \mathit{historyStateBinding} \in \mathit{HashType} : \\
\quad\quad \mathit{ValidateMAC}(\mathit{LL1NVRAM.symmetricKey},\ \mathit{historyStateBinding},\ \mathit{LL1RAM.authenticator}) \Rightarrow \\
\quad\quad\quad \mathit{LL1RAM.authenticator} \in \mathit{LL1ObservedAuthenticators}
\end{array}
$$

The *InclusionInvariant* states that, for any history summary and input that together hash to the history summary in the *NVRAM*, if this history summary is bound to some public and private state by an authenticated history state binding, then the result of invoking the service with this public state, private state, and input will yield an output that is already in the set of observed outputs and a new history state binding that is already authenticated.

This invariant is needed to show that the *LL1RepeatOperation* action does not have any ill effects. In particular, this invariant tells us that the output that *LL1RepeatOperation* will produce is already in *LL1ObservedOutputs*, and the new authenticator that *LL1RepeatOperation* will produce makes an assertion that is already being asserted by some authenticator in *LL1ObservedAuthenticators*.

$$
\begin{array}{l}
\mathit{InclusionInvariant} \triangleq \\
\quad \forall\ \mathit{input} \in \mathit{InputType}, \\
\quad\quad \mathit{historySummary} \in \mathit{HashType}, \\
\quad\quad \mathit{publicState} \in \mathit{PublicStateType}, \\
\quad\quad \mathit{privateStateEnc} \in \mathit{PrivateStateEncType} : \\
\quad\quad \text{LET} \\
\quad\quad\quad \mathit{stateHash} \triangleq \mathit{Hash}(\mathit{publicState},\ \mathit{privateStateEnc}) \\
\quad\quad\quad \mathit{historyStateBinding} \triangleq \mathit{Hash}(\mathit{historySummary},\ \mathit{stateHash}) \\
\quad\quad\quad \mathit{privateState} \triangleq \mathit{SymmetricDecrypt}(\mathit{LL1NVRAM.symmetricKey},\ \mathit{privateStateEnc}) \\
\quad\quad\quad \mathit{sResult} \triangleq \mathit{Service}(\mathit{publicState},\ \mathit{privateState},\ \mathit{input}) \\
\quad\quad\quad \mathit{newPrivateStateEnc} \triangleq \\
\quad\quad\quad\quad \mathit{SymmetricEncrypt}(\mathit{LL1NVRAM.symmetricKey},\ \mathit{sResult.newPrivateState}) \\
\quad\quad\quad \mathit{newStateHash} \triangleq \mathit{Hash}(\mathit{sResult.newPublicState},\ \mathit{newPrivateStateEnc}) \\
\quad\quad\quad \mathit{newHistoryStateBinding} \triangleq \mathit{Hash}(\mathit{LL1NVRAM.historySummary},\ \mathit{newStateHash}) \\
\quad\quad \text{IN} \\
\quad\quad\quad (\land\ \mathit{LL1NVRAM.historySummary} = \mathit{Hash}(\mathit{historySummary},\ \mathit{input}) \\
\quad\quad\quad\ \ \land\ \mathit{LL1HistoryStateBindingAuthenticated}(\mathit{historyStateBinding})) \\
\quad\quad\quad \Rightarrow \\
\quad\quad\quad (\land\ \mathit{sResult.output} \in \mathit{LL1ObservedOutputs} \\
\quad\quad\quad\ \ \land\ \mathit{LL1HistoryStateBindingAuthenticated}(\mathit{newHistoryStateBinding}))
\end{array}
$$

The *UniquenessInvariant* states that the the history summary in the *NVRAM* is bound to only one public and private state by an authenticator in the set of observed authenticators. This invariant is used in several places in the implementation proof.

In the *NonAdvancementLemma*, the property of uniqueness is needed to show that the refined public and private state does not change when the *NVRAM* and set of observed authenticators does not change. If there were more than one state bound to the history summary in the *NVRAM*, a low-level stuttering step could lead to a high-level change in state.

In the base case of the Memoir-Basic implementation proof, the uniqueness property is needed to show that the refined state hash corresponds uniquely to the initial state hash.

Within the induction of the Memoir-Basic implementation proof, in the case of *LL1PerformOperation*, the uniqueness property is used in two places. First, it is needed to show that the public and private state in the arguments to the service correspond to the refined high-level state. Second, it is needed to show that the public and private state produced as the result from the service correspond to the refined high-level primed state.

$$
\begin{array}{l}
\mathit{UniquenessInvariant} \triangleq \\
\quad \forall\, \mathit{stateHash}1,\ \mathit{stateHash}2 \in \mathit{HashType} : \\
\qquad \text{LET} \\
\qquad\quad \mathit{historyStateBinding}1 \triangleq \mathit{Hash}(\mathit{LL}1\mathit{NVRAM}.\mathit{historySummary},\ \mathit{stateHash}1) \\
\qquad\quad \mathit{historyStateBinding}2 \triangleq \mathit{Hash}(\mathit{LL}1\mathit{NVRAM}.\mathit{historySummary},\ \mathit{stateHash}2) \\
\qquad \text{IN} \\
\qquad\quad (\land\ \mathit{LL}1\mathit{HistoryStateBindingAuthenticated}(\mathit{historyStateBinding}1) \\
\qquad\quad\ \ \land\ \mathit{LL}1\mathit{HistoryStateBindingAuthenticated}(\mathit{historyStateBinding}2)) \\
\qquad\quad \Rightarrow \\
\qquad\qquad \mathit{stateHash}1 = \mathit{stateHash}2
\end{array}
$$

Collectively, we refer to these three invariants as the correctness invariants for the Memoir-Basic implementation.

$$
\begin{array}{l}
\mathit{CorrectnessInvariants} \triangleq \\
\quad \land\ \mathit{UnforgeabilityInvariant} \\
\quad \land\ \mathit{InclusionInvariant} \\
\quad \land\ \mathit{UniquenessInvariant}
\end{array}
$$

3.4 Invariants Needed to Prove Memoir-Basic Invariance

MODULE *MemoirLL1SupplementalInvariants*

This module defines two supplemental invariants. These are not needed directly by the Memoir-Basic implementation proof, but they are needed by the proofs that the correctness invariants hold.

EXTENDS $MemoirLL1CorrectnessInvariants$, $Naturals$

The *ExtendedUnforgeabilityInvariant* states that the unforgeability property of the authenticator in the RAM also applies to the authenticator on the disk. This is needed to show that the *UnforgeabilityInvariant* holds through a *LL1ReadDisk* action.

$$
\begin{array}{l}
ExtendedUnforgeabilityInvariant \triangleq \\
\quad \forall\, historyStateBinding \in HashType : \\
\qquad \land\ ValidateMAC(LL1NVRAM.symmetricKey,\ historyStateBinding,\ LL1RAM.authenticator) \Rightarrow \\
\qquad\qquad LL1RAM.authenticator \in LL1ObservedAuthenticators \\
\qquad \land\ ValidateMAC(LL1NVRAM.symmetricKey,\ historyStateBinding,\ LL1Disk.authenticator) \Rightarrow \\
\qquad\qquad LL1Disk.authenticator \in LL1ObservedAuthenticators
\end{array}
$$

The *CardinalityInvariant* is a little funky. We define a new operatior called *HashCardinality* that indicates the count of hashes needed to produce the supplied hash value. For example, if you create a hash chain, starting with the base hash value, and chain in N inputs, the cardinality of the resulting hash will be N. Although our low-level spec uses the hash operator only for linear hash chains, the hash cardinality is also defined for arbitrary trees of hashes.

The *CardinalityInvariant* states that the hash cardinality of the history summary in any observed authenticator is less than or equal to the hash cardinality of the history summary in the $NVRAM$.

This property is needed to prove that the uniqueness property is an inductive invariant.

CONSTANT $HashCardinality(\_)$

$$
\begin{array}{l}
\text{ASSUME } HashCardinalityTypeSafe \triangleq \\
\quad \forall\, hash \in HashDomain : HashCardinality(hash) \in Nat
\end{array}
$$

$$
\text{ASSUME } BaseHashCardinalityZero \triangleq HashCardinality(BaseHashValue) = 0
$$

$$
\begin{array}{l}
\text{ASSUME } InputCardinalityZero \triangleq \\
\quad \forall\, input \in InputType : HashCardinality(input) = 0
\end{array}
$$

$$
\begin{array}{l}
\text{ASSUME } HashCardinalityAccumulative \triangleq \\
\quad \forall\, hash1,\ hash2 \in HashDomain : \\
\qquad HashCardinality(Hash(hash1,\ hash2)) = \\
\qquad\quad HashCardinality(hash1) + HashCardinality(hash2) + 1
\end{array}
$$

$$
\begin{array}{l}
CardinalityInvariant \triangleq \\
\quad \forall\, historySummary \in HashType,\ stateHash \in HashType : \\
\qquad \text{LET} \\
\qquad\quad historyStateBinding \triangleq Hash(historySummary,\ stateHash) \\
\qquad \text{IN} \\
\qquad\quad (\land\ LL1NVRAMHistorySummaryUncorrupted \\
\qquad\quad\ \ \land\ LL1HistoryStateBindingAuthenticated(historyStateBinding)) \\
\qquad\ \Rightarrow \\
\qquad\quad HashCardinality(historySummary) \leq HashCardinality(LL1NVRAM.historySummary)
\end{array}
$$

4. PROOFS

This section presents TLA+ modules that contain proofs. The proofs include type safety of the three specs; lemmas relating to types, invariants, refinement, or implementation; proofs of invariance; and proofs of implementation.

4.1 Proof of Type Safety of the High-Level Spec

MODULE *MemoirHLTypeSafety*

This is a very simple proof that shows the high-level spec to be type-safe.

EXTENDS *MemoirHLSpecification*

THEOREM $HLTypeSafe \triangleq HLSpec \Rightarrow \Box HLTypeInvariant$

The top level of the proof is boilerplate TLA+ for an *Inv*1-style proof. First, we prove that the initial state satisfies *HLTypeInvariant*. Second, we prove that the *HLNext* predicate inductively preserves *HLTypeInvariant*. Third, we use temporal induction to prove that these two conditions satisfy type safety over all behaviors.

$\langle 1 \rangle 1.$ $HLInit \Rightarrow HLTypeInvariant$

The base case follows trivially from the definition of *HLInit* and the assumption that the developer-supplied constants are type-safe.

$\langle 2 \rangle 1.$ HAVE *HLInit*

$\langle 2 \rangle 2.$ QED

BY $\langle 2 \rangle 1$, *ConstantsTypeSafe*DEF *HLInit*, *HLTypeInvariant*

$\langle 1 \rangle 2.$ $HLTypeInvariant \land [HLNext]_{HLVars} \Rightarrow HLTypeInvariant'$

The induction step is also fairly trivial. We assume the antecedents of the implication, then show that the consequent holds for both *HLNext* actions.

$\langle 2 \rangle 1.$ HAVE $HLTypeInvariant \land [HLNext]_{HLVars}$

$\langle 2 \rangle 2.$ CASE UNCHANGED *HLVars*

Type safety is inductively trivial for a stuttering step.

BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$ DEF *HLTypeInvariant*, *HLVars*

$\langle 2 \rangle 3.$ CASE *HLMakeInputAvailable*

Type safety is also trivial for a *HLMakeInputAvailable* action.

BY $\langle 2 \rangle 1$, $\langle 2 \rangle 3$ DEF *HLTypeInvariant*, *HLMakeInputAvailable*

$\langle 2 \rangle 4.$ CASE *HLAdvanceService*

For a *HLAdvanceService* action, we just walk through the definitions. Type safety follows directly.

$\langle 3 \rangle 1.$ PICK $input \in HLAvailableInputs : HLAdvanceService!(input)!1$

BY DEF *HLAdvanceService*

$\langle 3 \rangle 2.$ $\land\ HLPublicState \in PublicStateType$
$\land\ HLPrivateState \in PrivateStateType$
$\land\ HLAvailableInputs \subseteq InputType$

BY $\langle 2 \rangle 1$ DEF *HLTypeInvariant*

$\langle 3 \rangle 3.$ $HLAdvanceService!(input)!hlSResult \in ServiceResultType$

BY $\langle 3 \rangle 1$, $\langle 3 \rangle 2$, *ServiceTypeSafe*

$\langle 3 \rangle 4.$ $\land\ HLAdvanceService!(input)!hlSResult.newPublicState \in PublicStateType$
$\land\ HLAdvanceService!(input)!hlSResult.newPrivateState \in PrivateStateType$
$\land\ HLAdvanceService!(input)!hlSResult.output \in OutputType$

BY $\langle 3 \rangle 3$ DEF *ServiceResultType*

$\langle 3 \rangle 5.$ QED

BY $\langle 2 \rangle 1$, $\langle 3 \rangle 1$, $\langle 3 \rangle 4$ DEF *HLAdvanceService*, *HLTypeInvariant*

$\langle 2 \rangle 5.$ CASE *HLDie*

Type safety is also trivial for a *HLDie* action.

BY $\langle 2 \rangle 1$, $\langle 2 \rangle 5$, *ConstantsTypeSafe*DEF *HLTypeInvariant*, *HLDie*

$\langle 2 \rangle 6.$ QED

BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, $\langle 2 \rangle 3$, $\langle 2 \rangle 4$, $\langle 2 \rangle 5$ DEF *HLNext*

$\langle 1 \rangle 3.$ QED

Using the *Inv*1 proof rule, the base case and the induction step together imply that the invariant always holds.

$\langle 2 \rangle 1.$ $HLTypeInvariant \land \Box[HLNext]_{HLVars} \Rightarrow \Box HLTypeInvariant$

BY $\langle 1 \rangle 2$, $Inv1$

$\langle 2 \rangle 2$. QED

BY $\langle 2 \rangle 1$, $\langle 1 \rangle 1$ DEF $HLSpec$

4.2 Proofs of Lemmas Relating to Types in the Memoir-Basic Spec

MODULE *MemoirLL1TypeLemmas*

This module states and proves sevaral lemmas that are useful for proving type safety. Since type safety is an important part of the implementation proof, these lemmas also will be used in theorems other than the Memoir-Basic type-safety theorem.

The lemmas in this module are:
LL1SubtypeImplicationLemma
LL1InitDefsTypeSafeLemma
LL1PerformOperationDefsTypeSafeLemma
LL1RepeatOperationDefsTypeSafeLemma
InclusionInvariantDefsTypeSafeLemma
CardinalityInvariantDefsTypeSafeLemma
UniquenessInvariantDefsTypeSafeLemma
LL1NVRAMHistorySummaryUncorruptedDefsTypeSafeLemma
LL1RefinementDefsTypeSafeLemma
LL1RefinementPrimeDefsTypeSafeLemma

EXTENDS *MemoirLL1Refinement*

LL1SubtypeImplicationLemma proves that when the *LL1TypeInvariant* holds, the subtypes of *LL1Disk*, *LL1RAM*, and *LL1NVRAM* also hold. This is asserted and proven for both the unprimed and primed states.

The proof itself is completely trivial. It follows directly from the type definitions *LL1UntrustedStorageType* and *LL1TrustedStorageType*.

$LL1SubtypeImplication \triangleq$
$\quad LL1TypeInvariant \Rightarrow$
$\quad\quad \land\ LL1Disk.publicState \in PublicStateType$
$\quad\quad \land\ LL1Disk.privateStateEnc \in PrivateStateEncType$
$\quad\quad \land\ LL1Disk.historySummary \in HashType$
$\quad\quad \land\ LL1Disk.authenticator \in MACType$
$\quad\quad \land\ LL1RAM.publicState \in PublicStateType$
$\quad\quad \land\ LL1RAM.privateStateEnc \in PrivateStateEncType$
$\quad\quad \land\ LL1RAM.historySummary \in HashType$
$\quad\quad \land\ LL1RAM.authenticator \in MACType$
$\quad\quad \land\ LL1NVRAM.historySummary \in HashType$
$\quad\quad \land\ LL1NVRAM.symmetricKey \in SymmetricKeyType$

THEOREM $LL1SubtypeImplicationLemma \triangleq$
$\quad \land LL1SubtypeImplication$
$\quad \land LL1SubtypeImplication'$
$\langle 1 \rangle 1.\ LL1SubtypeImplication$
$\quad \langle 2 \rangle 1.$ SUFFICES
$\quad\quad$ ASSUME $LL1TypeInvariant$
$\quad\quad$ PROVE
$\quad\quad\quad \land\ LL1Disk.publicState \in PublicStateType$
$\quad\quad\quad \land\ LL1Disk.privateStateEnc \in PrivateStateEncType$
$\quad\quad\quad \land\ LL1Disk.historySummary \in HashType$
$\quad\quad\quad \land\ LL1Disk.authenticator \in MACType$
$\quad\quad\quad \land\ LL1RAM.publicState \in PublicStateType$
$\quad\quad\quad \land\ LL1RAM.privateStateEnc \in PrivateStateEncType$
$\quad\quad\quad \land\ LL1RAM.historySummary \in HashType$
$\quad\quad\quad \land\ LL1RAM.authenticator \in MACType$
$\quad\quad\quad \land\ LL1NVRAM.historySummary \in HashType$

$\wedge$ $LL1NVRAM.symmetricKey \in SymmetricKeyType$

BY DEF $LL1SubtypeImplication$

$\langle 2 \rangle 2$. $LL1Disk \in LL1UntrustedStorageType$

BY $\langle 2 \rangle 1$ DEF $LL1TypeInvariant$

$\langle 2 \rangle 3$. $LL1RAM \in LL1UntrustedStorageType$

BY $\langle 2 \rangle 1$ DEF $LL1TypeInvariant$

$\langle 2 \rangle 4$. $LL1NVRAM \in LL1TrustedStorageType$

BY $\langle 2 \rangle 1$ DEF $LL1TypeInvariant$

$\langle 2 \rangle 5$. $LL1Disk.publicState \in PublicStateType$

BY $\langle 2 \rangle 2$ DEF $LL1UntrustedStorageType$

$\langle 2 \rangle 6$. $LL1Disk.privateStateEnc \in PrivateStateEncType$

BY $\langle 2 \rangle 2$ DEF $LL1UntrustedStorageType$

$\langle 2 \rangle 7$. $LL1Disk.historySummary \in HashType$

BY $\langle 2 \rangle 2$ DEF $LL1UntrustedStorageType$

$\langle 2 \rangle 8$. $LL1Disk.authenticator \in MACType$

BY $\langle 2 \rangle 2$ DEF $LL1UntrustedStorageType$

$\langle 2 \rangle 9$. $LL1RAM.publicState \in PublicStateType$

BY $\langle 2 \rangle 3$ DEF $LL1UntrustedStorageType$

$\langle 2 \rangle 10$. $LL1RAM.privateStateEnc \in PrivateStateEncType$

BY $\langle 2 \rangle 3$ DEF $LL1UntrustedStorageType$

$\langle 2 \rangle 11$. $LL1RAM.historySummary \in HashType$

BY $\langle 2 \rangle 3$ DEF $LL1UntrustedStorageType$

$\langle 2 \rangle 12$. $LL1RAM.authenticator \in MACType$

BY $\langle 2 \rangle 3$ DEF $LL1UntrustedStorageType$

$\langle 2 \rangle 13$. $LL1NVRAM.historySummary \in HashType$

BY $\langle 2 \rangle 4$ DEF $LL1TrustedStorageType$

$\langle 2 \rangle 14$. $LL1NVRAM.symmetricKey \in SymmetricKeyType$

BY $\langle 2 \rangle 4$ DEF $LL1TrustedStorageType$

$\langle 2 \rangle 15$. QED

BY $\langle 2 \rangle 5$, $\langle 2 \rangle 6$, $\langle 2 \rangle 7$, $\langle 2 \rangle 8$, $\langle 2 \rangle 9$, $\langle 2 \rangle 10$, $\langle 2 \rangle 11$, $\langle 2 \rangle 12$, $\langle 2 \rangle 13$, $\langle 2 \rangle 14$

$\langle 1 \rangle 2$. $LL1SubtypeImplication'$

$\langle 2 \rangle 1$. SUFFICES

ASSUME $LL1TypeInvariant'$

PROVE

$\wedge$ $LL1Disk.publicState' \in PublicStateType$

$\wedge$ $LL1Disk.privateStateEnc' \in PrivateStateEncType$

$\wedge$ $LL1Disk.historySummary' \in HashType$

$\wedge$ $LL1Disk.authenticator' \in MACType$

$\wedge$ $LL1RAM.publicState' \in PublicStateType$

$\wedge$ $LL1RAM.privateStateEnc' \in PrivateStateEncType$

$\wedge$ $LL1RAM.historySummary' \in HashType$

$\wedge$ $LL1RAM.authenticator' \in MACType$

$\wedge$ $LL1NVRAM.historySummary' \in HashType$

$\wedge$ $LL1NVRAM.symmetricKey' \in SymmetricKeyType$

BY DEF $LL1SubtypeImplication$

$\langle 2 \rangle 2$. $LL1Disk' \in LL1UntrustedStorageType$

BY $\langle 2 \rangle 1$ DEF $LL1TypeInvariant$

$\langle 2 \rangle 3$. $LL1RAM' \in LL1UntrustedStorageType$

BY $\langle 2 \rangle 1$ DEF $LL1TypeInvariant$

$\langle 2 \rangle 4$. $LL1NVRAM' \in LL1TrustedStorageType$

BY $\langle 2 \rangle 1$ DEF $LL1TypeInvariant$

$\langle 2 \rangle 5$. $LL1Disk.publicState' \in PublicStateType$

BY $\langle 2\rangle 2$ DEF *LL1UntrustedStorageType*
$\langle 2\rangle 6.$ $LL1Disk.privateStateEnc' \in PrivateStateEncType$
BY $\langle 2\rangle 2$ DEF *LL1UntrustedStorageType*
$\langle 2\rangle 7.$ $LL1Disk.historySummary' \in HashType$
BY $\langle 2\rangle 2$ DEF *LL1UntrustedStorageType*
$\langle 2\rangle 8.$ $LL1Disk.authenticator' \in MACType$
BY $\langle 2\rangle 2$ DEF *LL1UntrustedStorageType*
$\langle 2\rangle 9.$ $LL1RAM.publicState' \in PublicStateType$
BY $\langle 2\rangle 3$ DEF *LL1UntrustedStorageType*
$\langle 2\rangle 10.$ $LL1RAM.privateStateEnc' \in PrivateStateEncType$
BY $\langle 2\rangle 3$ DEF *LL1UntrustedStorageType*
$\langle 2\rangle 11.$ $LL1RAM.historySummary' \in HashType$
BY $\langle 2\rangle 3$ DEF *LL1UntrustedStorageType*
$\langle 2\rangle 12.$ $LL1RAM.authenticator' \in MACType$
BY $\langle 2\rangle 3$ DEF *LL1UntrustedStorageType*
$\langle 2\rangle 13.$ $LL1NVRAM.historySummary' \in HashType$
BY $\langle 2\rangle 4$ DEF *LL1TrustedStorageType*
$\langle 2\rangle 14.$ $LL1NVRAM.symmetricKey' \in SymmetricKeyType$
BY $\langle 2\rangle 4$ DEF *LL1TrustedStorageType*
$\langle 2\rangle 15.$ QED
BY $\langle 2\rangle 5$, $\langle 2\rangle 6$, $\langle 2\rangle 7$, $\langle 2\rangle 8$, $\langle 2\rangle 9$, $\langle 2\rangle 10$, $\langle 2\rangle 11$, $\langle 2\rangle 12$, $\langle 2\rangle 13$, $\langle 2\rangle 14$
$\langle 1\rangle 3.$ QED
BY $\langle 1\rangle 1$, $\langle 1\rangle 2$

LL1InitDefsTypeSafeLemma proves that the definitions within the LET of the *LL1Init* action all have the appropriate type. This is a trivial proof that merely walks through the definitions.

THEOREM $LL1InitDefsTypeSafeLemma \triangleq$
$\forall\, symmetricKey \in SymmetricKeyType :$
LET
$initialPrivateStateEnc \triangleq SymmetricEncrypt(symmetricKey, InitialPrivateState)$
$initialStateHash \triangleq Hash(InitialPublicState, initialPrivateStateEnc)$
$initialHistoryStateBinding \triangleq Hash(BaseHashValue, initialStateHash)$
$initialAuthenticator \triangleq GenerateMAC(symmetricKey, initialHistoryStateBinding)$
$initialUntrustedStorage \triangleq [$
$publicState \mapsto InitialPublicState,$
$privateStateEnc \mapsto initialPrivateStateEnc,$
$historySummary \mapsto BaseHashValue,$
$authenticator \mapsto initialAuthenticator]$
$initialTrustedStorage \triangleq [$
$historySummary \mapsto BaseHashValue,$
$symmetricKey \mapsto symmetricKey]$
IN
$\land\ initialPrivateStateEnc \in PrivateStateEncType$
$\land\ initialStateHash \in HashType$
$\land\ initialHistoryStateBinding \in HashType$
$\land\ initialAuthenticator \in MACType$
$\land\ initialUntrustedStorage \in LL1UntrustedStorageType$
$\land\ initialTrustedStorage \in LL1TrustedStorageType$
$\langle 1\rangle 1.$ TAKE $symmetricKey \in SymmetricKeyType$
$\langle 1\rangle$ $initialPrivateStateEnc \triangleq SymmetricEncrypt(symmetricKey, InitialPrivateState)$
$\langle 1\rangle$ $initialStateHash \triangleq Hash(InitialPublicState, initialPrivateStateEnc)$
$\langle 1\rangle$ $initialHistoryStateBinding \triangleq Hash(BaseHashValue, initialStateHash)$

$\langle 1 \rangle$ $initialAuthenticator \triangleq GenerateMAC(symmetricKey, initialHistoryStateBinding)$
$\langle 1 \rangle$ $initialUntrustedStorage \triangleq [$
$publicState \mapsto InitialPublicState,$
$privateStateEnc \mapsto initialPrivateStateEnc,$
$historySummary \mapsto BaseHashValue,$
$authenticator \mapsto initialAuthenticator]$
$\langle 1 \rangle$ $initialTrustedStorage \triangleq [$
$historySummary \mapsto BaseHashValue,$
$symmetricKey \mapsto symmetricKey]$
$\langle 1 \rangle$ HIDE DEF $initialPrivateStateEnc$, $initialStateHash$, $initialAuthenticator$, $initialUntrustedStorage$, $initialTrustedStorage$
$\langle 1 \rangle 2.$ $initialPrivateStateEnc \in PrivateStateEncType$
$\langle 2 \rangle 1.$ $symmetricKey \in SymmetricKeyType$
BY $\langle 1 \rangle 1$
$\langle 2 \rangle 2.$ $InitialPrivateState \in PrivateStateType$
BY $ConstantsTypeSafe$
$\langle 2 \rangle 3.$ QED
BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, $SymmetricEncryptionTypeSafe$DEF $initialPrivateStateEnc$
$\langle 1 \rangle 3.$ $initialStateHash \in HashType$
$\langle 2 \rangle 1.$ $InitialPublicState \in HashDomain$
$\langle 3 \rangle 1.$ $InitialPublicState \in PublicStateType$
BY $ConstantsTypeSafe$
$\langle 3 \rangle 2.$ QED
BY $\langle 3 \rangle 1$ DEF $HashDomain$
$\langle 2 \rangle 2.$ $initialPrivateStateEnc \in HashDomain$
BY $\langle 1 \rangle 2$ DEF $HashDomain$
$\langle 2 \rangle 3.$ QED
BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, $HashTypeSafe$DEF $initialStateHash$
$\langle 1 \rangle 4.$ $initialHistoryStateBinding \in HashType$
$\langle 2 \rangle 1.$ $BaseHashValue \in HashDomain$
$\langle 3 \rangle 1.$ $BaseHashValue \in HashType$
BY $BaseHashValueTypeSafe$
$\langle 3 \rangle 2.$ QED
BY $\langle 3 \rangle 1$ DEF $HashDomain$
$\langle 2 \rangle 2.$ $initialStateHash \in HashDomain$
BY $\langle 1 \rangle 3$ DEF $HashDomain$
$\langle 2 \rangle 3.$ QED
BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, $HashTypeSafe$DEF $initialHistoryStateBinding$
$\langle 1 \rangle 5.$ $initialAuthenticator \in MACType$
$\langle 2 \rangle 1.$ $symmetricKey \in SymmetricKeyType$
BY $\langle 1 \rangle 1$
$\langle 2 \rangle 2.$ $initialHistoryStateBinding \in HashType$
BY $\langle 1 \rangle 4$
$\langle 2 \rangle 3.$ QED
BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, $GenerateMACTypeSafe$DEF $initialAuthenticator$
$\langle 1 \rangle 6.$ $initialUntrustedStorage \in LL1UntrustedStorageType$
$\langle 2 \rangle 1.$ $InitialPublicState \in PublicStateType$
BY $ConstantsTypeSafe$
$\langle 2 \rangle 2.$ $initialPrivateStateEnc \in PrivateStateEncType$
BY $\langle 1 \rangle 2$
$\langle 2 \rangle 3.$ $BaseHashValue \in HashType$
BY $BaseHashValueTypeSafe$

$\langle 2\rangle 4.\ \mathit{initialAuthenticator} \in \mathit{MACType}$
BY $\langle 1\rangle 5$
$\langle 2\rangle 5.$ QED
BY $\langle 2\rangle 1$, $\langle 2\rangle 2$, $\langle 2\rangle 3$, $\langle 2\rangle 4$ DEF $\mathit{initialUntrustedStorage}$, $\mathit{LL1UntrustedStorageType}$
$\langle 1\rangle 7.\ \mathit{initialTrustedStorage} \in \mathit{LL1TrustedStorageType}$
$\langle 2\rangle 1.\ \mathit{BaseHashValue} \in \mathit{HashType}$
BY $\mathit{BaseHashValueTypeSafe}$
$\langle 2\rangle 2.\ \mathit{symmetricKey} \in \mathit{SymmetricKeyType}$
BY $\langle 1\rangle 1$
$\langle 2\rangle 3.$ QED
BY $\langle 2\rangle 1$, $\langle 2\rangle 2$ DEF $\mathit{initialTrustedStorage}$, $\mathit{LL1TrustedStorageType}$
$\langle 1\rangle 8.$ QED
BY $\langle 1\rangle 2$, $\langle 1\rangle 3$, $\langle 1\rangle 4$, $\langle 1\rangle 5$, $\langle 1\rangle 6$, $\langle 1\rangle 7$
DEF $\mathit{initialPrivateStateEnc}$, $\mathit{initialStateHash}$, $\mathit{initialAuthenticator}$,
$\mathit{initialUntrustedStorage}$, $\mathit{initialTrustedStorage}$

LL1PerformOperationDefsTypeSafeLemma proves that the definitions within the LET of the *LL1PerformOperation* action all have the appropriate type. This is a trivial proof that merely walks through the definitions.

THEOREM $\mathit{LL1PerformOperationDefsTypeSafeLemma} \triangleq$
$\forall\, \mathit{input} \in \mathit{LL1AvailableInputs}:$
$\mathit{LL1TypeInvariant} \Rightarrow$
LET
$\mathit{stateHash} \triangleq \mathit{Hash}(\mathit{LL1RAM.publicState}, \mathit{LL1RAM.privateStateEnc})$
$\mathit{historyStateBinding} \triangleq \mathit{Hash}(\mathit{LL1RAM.historySummary}, \mathit{stateHash})$
$\mathit{privateState} \triangleq \mathit{SymmetricDecrypt}(\mathit{LL1NVRAM.symmetricKey}, \mathit{LL1RAM.privateStateEnc})$
$\mathit{sResult} \triangleq \mathit{Service}(\mathit{LL1RAM.publicState}, \mathit{privateState}, \mathit{input})$
$\mathit{newPrivateStateEnc} \triangleq$
$\mathit{SymmetricEncrypt}(\mathit{LL1NVRAM.symmetricKey}, \mathit{sResult.newPrivateState})$
$\mathit{newHistorySummary} \triangleq \mathit{Hash}(\mathit{LL1NVRAM.historySummary}, \mathit{input})$
$\mathit{newStateHash} \triangleq \mathit{Hash}(\mathit{sResult.newPublicState}, \mathit{newPrivateStateEnc})$
$\mathit{newHistoryStateBinding} \triangleq \mathit{Hash}(\mathit{newHistorySummary}, \mathit{newStateHash})$
$\mathit{newAuthenticator} \triangleq \mathit{GenerateMAC}(\mathit{LL1NVRAM.symmetricKey}, \mathit{newHistoryStateBinding})$
IN
$\land\ \mathit{stateHash} \in \mathit{HashType}$
$\land\ \mathit{historyStateBinding} \in \mathit{HashType}$
$\land\ \mathit{privateState} \in \mathit{PrivateStateType}$
$\land\ \mathit{sResult} \in \mathit{ServiceResultType}$
$\land\ \mathit{sResult.newPublicState} \in \mathit{PublicStateType}$
$\land\ \mathit{sResult.newPrivateState} \in \mathit{PrivateStateType}$
$\land\ \mathit{sResult.output} \in \mathit{OutputType}$
$\land\ \mathit{newPrivateStateEnc} \in \mathit{PrivateStateEncType}$
$\land\ \mathit{newHistorySummary} \in \mathit{HashType}$
$\land\ \mathit{newStateHash} \in \mathit{HashType}$
$\land\ \mathit{newHistoryStateBinding} \in \mathit{HashType}$
$\land\ \mathit{newAuthenticator} \in \mathit{MACType}$
$\langle 1\rangle 1.$ TAKE $\mathit{input} \in \mathit{LL1AvailableInputs}$
$\langle 1\rangle\ \mathit{stateHash} \triangleq \mathit{Hash}(\mathit{LL1RAM.publicState}, \mathit{LL1RAM.privateStateEnc})$
$\langle 1\rangle\ \mathit{historyStateBinding} \triangleq \mathit{Hash}(\mathit{LL1RAM.historySummary}, \mathit{stateHash})$
$\langle 1\rangle\ \mathit{privateState} \triangleq \mathit{SymmetricDecrypt}(\mathit{LL1NVRAM.symmetricKey}, \mathit{LL1RAM.privateStateEnc})$
$\langle 1\rangle\ \mathit{sResult} \triangleq \mathit{Service}(\mathit{LL1RAM.publicState}, \mathit{privateState}, \mathit{input})$
$\langle 1\rangle\ \mathit{newPrivateStateEnc} \triangleq$
$\mathit{SymmetricEncrypt}(\mathit{LL1NVRAM.symmetricKey}, \mathit{sResult.newPrivateState})$

$\langle 1 \rangle$ $\mathit{newHistorySummary} \triangleq \mathit{Hash}(\mathit{LL1NVRAM.historySummary}, \mathit{input})$
$\langle 1 \rangle$ $\mathit{newStateHash} \triangleq \mathit{Hash}(\mathit{sResult.newPublicState}, \mathit{newPrivateStateEnc})$
$\langle 1 \rangle$ $\mathit{newHistoryStateBinding} \triangleq \mathit{Hash}(\mathit{newHistorySummary}, \mathit{newStateHash})$
$\langle 1 \rangle$ $\mathit{newAuthenticator} \triangleq \mathit{GenerateMAC}(\mathit{LL1NVRAM.symmetricKey}, \mathit{newHistoryStateBinding})$
$\langle 1 \rangle$ HIDE DEF $\mathit{stateHash}$, $\mathit{historyStateBinding}$, $\mathit{privateState}$, $\mathit{sResult}$, $\mathit{newPrivateStateEnc}$,
$\mathit{newHistorySummary}$, $\mathit{newStateHash}$, $\mathit{newHistoryStateBinding}$, $\mathit{newAuthenticator}$
$\langle 1 \rangle 2.$ HAVE $\mathit{LL1TypeInvariant}$
$\langle 1 \rangle 3.$ $\mathit{stateHash} \in \mathit{HashType}$
$\langle 2 \rangle 1.$ $\land\ \mathit{LL1RAM.publicState} \in \mathit{PublicStateType}$
$\land\ \mathit{LL1RAM.privateStateEnc} \in \mathit{PrivateStateEncType}$
BY $\langle 1 \rangle 2$, $\mathit{LL1SubtypeImplicationLemma}$DEF $\mathit{LL1SubtypeImplication}$
$\langle 2 \rangle 2.$ $\land\ \mathit{LL1RAM.publicState} \in \mathit{HashDomain}$
$\land\ \mathit{LL1RAM.privateStateEnc} \in \mathit{HashDomain}$
BY $\langle 2 \rangle 1$ DEF $\mathit{HashDomain}$
$\langle 2 \rangle 3.$ QED
BY $\langle 2 \rangle 2$, $\mathit{HashTypeSafe}$DEF $\mathit{stateHash}$
$\langle 1 \rangle 4.$ $\mathit{historyStateBinding} \in \mathit{HashType}$
$\langle 2 \rangle 1.$ $\mathit{LL1RAM.historySummary} \in \mathit{HashDomain}$
$\langle 3 \rangle 1.$ $\mathit{LL1RAM.historySummary} \in \mathit{HashType}$
BY $\langle 1 \rangle 2$, $\mathit{LL1SubtypeImplicationLemma}$DEF $\mathit{LL1SubtypeImplication}$
$\langle 3 \rangle 2.$ QED
BY $\langle 3 \rangle 1$ DEF $\mathit{HashDomain}$
$\langle 2 \rangle 2.$ $\mathit{stateHash} \in \mathit{HashDomain}$
BY $\langle 1 \rangle 3$ DEF $\mathit{HashDomain}$
$\langle 2 \rangle 3.$ QED
BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, $\mathit{HashTypeSafe}$DEF $\mathit{historyStateBinding}$
$\langle 1 \rangle 5.$ $\mathit{privateState} \in \mathit{PrivateStateType}$
$\langle 2 \rangle 1.$ $\land\ \mathit{LL1NVRAM.symmetricKey} \in \mathit{SymmetricKeyType}$
$\land\ \mathit{LL1RAM.privateStateEnc} \in \mathit{PrivateStateEncType}$
BY $\langle 1 \rangle 2$, $\mathit{LL1SubtypeImplicationLemma}$DEF $\mathit{LL1SubtypeImplication}$
$\langle 2 \rangle 2.$ QED
BY $\langle 2 \rangle 1$, $\mathit{SymmetricDecryptionTypeSafe}$DEF $\mathit{privateState}$
$\langle 1 \rangle 6.$ $\mathit{sResult} \in \mathit{ServiceResultType}$
$\langle 2 \rangle 1.$ $\mathit{LL1RAM.publicState} \in \mathit{PublicStateType}$
BY $\langle 1 \rangle 2$, $\mathit{LL1SubtypeImplicationLemma}$DEF $\mathit{LL1SubtypeImplication}$
$\langle 2 \rangle 2.$ $\mathit{privateState} \in \mathit{PrivateStateType}$
BY $\langle 1 \rangle 5$
$\langle 2 \rangle 3.$ $\mathit{input} \in \mathit{InputType}$
$\langle 3 \rangle 1.$ $\mathit{LL1AvailableInputs} \subseteq \mathit{InputType}$
BY $\langle 1 \rangle 2$ DEF $\mathit{LL1TypeInvariant}$
$\langle 3 \rangle 2.$ QED
BY $\langle 1 \rangle 1$, $\langle 3 \rangle 1$
$\langle 2 \rangle 4.$ QED
BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, $\langle 2 \rangle 3$, $\mathit{ServiceTypeSafe}$DEF $\mathit{sResult}$
$\langle 1 \rangle 7.$ $\land\ \mathit{sResult.newPublicState} \in \mathit{PublicStateType}$
$\land\ \mathit{sResult.newPrivateState} \in \mathit{PrivateStateType}$
$\land\ \mathit{sResult.output} \in \mathit{OutputType}$
BY $\langle 1 \rangle 6$ DEF $\mathit{ServiceResultType}$
$\langle 1 \rangle 8.$ $\mathit{newPrivateStateEnc} \in \mathit{PrivateStateEncType}$
$\langle 2 \rangle 1.$ $\mathit{LL1NVRAM.symmetricKey} \in \mathit{SymmetricKeyType}$
BY $\langle 1 \rangle 2$, $\mathit{LL1SubtypeImplicationLemma}$DEF $\mathit{LL1SubtypeImplication}$
$\langle 2 \rangle 2.$ $\mathit{sResult.newPrivateState} \in \mathit{PrivateStateType}$

BY $\langle 1 \rangle 7$

$\langle 2 \rangle 3.$ QED

BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, *SymmetricEncryptionTypeSafe*DEF *newPrivateStateEnc*

$\langle 1 \rangle 9.$ $newHistorySummary \in HashType$

$\langle 2 \rangle 1.$ $LL1NVRAM.historySummary \in HashDomain$

$\langle 3 \rangle 1.$ $LL1NVRAM.historySummary \in HashType$

BY $\langle 1 \rangle 2$, *LL1SubtypeImplicationLemma*DEF *LL1SubtypeImplication*

$\langle 3 \rangle 2.$ QED

BY $\langle 3 \rangle 1$ DEF *HashDomain*

$\langle 2 \rangle 2.$ $input \in HashDomain$

$\langle 3 \rangle 1.$ $input \in InputType$

$\langle 4 \rangle 1.$ $LL1AvailableInputs \subseteq InputType$

BY $\langle 1 \rangle 2$ DEF *LL1TypeInvariant*

$\langle 4 \rangle 2.$ QED

BY $\langle 1 \rangle 1$, $\langle 4 \rangle 1$

$\langle 3 \rangle 2.$ QED

BY $\langle 3 \rangle 1$ DEF *HashDomain*

$\langle 2 \rangle 3.$ QED

BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, *HashTypeSafe*DEF *newHistorySummary*

$\langle 1 \rangle 10.$ $newStateHash \in HashType$

$\langle 2 \rangle 1.$ $sResult.newPublicState \in HashDomain$

$\langle 3 \rangle 1.$ $sResult.newPublicState \in PublicStateType$

BY $\langle 1 \rangle 7$

$\langle 3 \rangle 2.$ QED

BY $\langle 3 \rangle 1$ DEF *HashDomain*

$\langle 2 \rangle 2.$ $newPrivateStateEnc \in HashDomain$

BY $\langle 1 \rangle 8$ DEF *HashDomain*

$\langle 2 \rangle 3.$ QED

BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, *HashTypeSafe*DEF *newStateHash*

$\langle 1 \rangle 11.$ $newHistoryStateBinding \in HashType$

$\langle 2 \rangle 1.$ $newHistorySummary \in HashDomain$

BY $\langle 1 \rangle 9$ DEF *HashDomain*

$\langle 2 \rangle 2.$ $newStateHash \in HashDomain$

BY $\langle 1 \rangle 10$ DEF *HashDomain*

$\langle 2 \rangle 3.$ QED

BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, *HashTypeSafe*DEF *newHistoryStateBinding*

$\langle 1 \rangle 12.$ $newAuthenticator \in MACType$

$\langle 2 \rangle 1.$ $LL1NVRAM.symmetricKey \in SymmetricKeyType$

BY $\langle 1 \rangle 2$, *LL1SubtypeImplicationLemma*DEF *LL1SubtypeImplication*

$\langle 2 \rangle 2.$ $newHistoryStateBinding \in HashType$

BY $\langle 1 \rangle 11$

$\langle 2 \rangle 3.$ QED

BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, *GenerateMACTypeSafe*DEF *newAuthenticator*

$\langle 1 \rangle 13.$ QED

BY $\langle 1 \rangle 3$, $\langle 1 \rangle 4$, $\langle 1 \rangle 5$, $\langle 1 \rangle 6$, $\langle 1 \rangle 7$, $\langle 1 \rangle 8$, $\langle 1 \rangle 9$, $\langle 1 \rangle 10$, $\langle 1 \rangle 11$, $\langle 1 \rangle 12$

DEF *stateHash*, *historyStateBinding*, *privateState*, *sResult*, *newPrivateStateEnc*, *newHistorySummary*, *newStateHash*, *newHistoryStateBinding*, *newAuthenticator*

LL1RepeatOperationDefsTypeSafeLemma proves that the definitions within the LET of the *LL1RepeatOperation* action all have the appropriate type. This is a trivial proof that merely walks through the definitions.

THEOREM *LL1RepeatOperationDefsTypeSafeLemma* $\triangleq$

$\forall\, input \in LL1AvailableInputs :$

$LL1TypeInvariant \Rightarrow$

LET

$stateHash \triangleq Hash(LL1RAM.publicState, LL1RAM.privateStateEnc)$
$historyStateBinding \triangleq Hash(LL1RAM.historySummary, stateHash)$
$privateState \triangleq SymmetricDecrypt(LL1NVRAM.symmetricKey, LL1RAM.privateStateEnc)$
$sResult \triangleq Service(LL1RAM.publicState, privateState, input)$
$newPrivateStateEnc \triangleq$
$SymmetricEncrypt(LL1NVRAM.symmetricKey, sResult.newPrivateState)$
$newStateHash \triangleq Hash(sResult.newPublicState, newPrivateStateEnc)$
$newHistoryStateBinding \triangleq Hash(LL1NVRAM.historySummary, newStateHash)$
$newAuthenticator \triangleq GenerateMAC(LL1NVRAM.symmetricKey, newHistoryStateBinding)$

IN

$\land\ stateHash \in HashType$
$\land\ historyStateBinding \in HashType$
$\land\ privateState \in PrivateStateType$
$\land\ sResult \in ServiceResultType$
$\land\ sResult.newPublicState \in PublicStateType$
$\land\ sResult.newPrivateState \in PrivateStateType$
$\land\ sResult.output \in OutputType$
$\land\ newPrivateStateEnc \in PrivateStateEncType$
$\land\ newStateHash \in HashType$
$\land\ newHistoryStateBinding \in HashType$
$\land\ newAuthenticator \in MACType$

$\langle 1 \rangle 1.$ TAKE $input \in LL1AvailableInputs$
$\langle 1 \rangle$ $stateHash \triangleq Hash(LL1RAM.publicState, LL1RAM.privateStateEnc)$
$\langle 1 \rangle$ $historyStateBinding \triangleq Hash(LL1RAM.historySummary, stateHash)$
$\langle 1 \rangle$ $privateState \triangleq SymmetricDecrypt(LL1NVRAM.symmetricKey, LL1RAM.privateStateEnc)$
$\langle 1 \rangle$ $sResult \triangleq Service(LL1RAM.publicState, privateState, input)$
$\langle 1 \rangle$ $newPrivateStateEnc \triangleq$
$SymmetricEncrypt(LL1NVRAM.symmetricKey, sResult.newPrivateState)$
$\langle 1 \rangle$ $newStateHash \triangleq Hash(sResult.newPublicState, newPrivateStateEnc)$
$\langle 1 \rangle$ $newHistoryStateBinding \triangleq Hash(LL1NVRAM.historySummary, newStateHash)$
$\langle 1 \rangle$ $newAuthenticator \triangleq GenerateMAC(LL1NVRAM.symmetricKey, newHistoryStateBinding)$
$\langle 1 \rangle$ HIDE DEF $stateHash$, $historyStateBinding$, $privateState$, $sResult$, $newPrivateStateEnc$,
$newStateHash$, $newHistoryStateBinding$, $newAuthenticator$
$\langle 1 \rangle 2.$ HAVE $LL1TypeInvariant$
$\langle 1 \rangle 3.$ $stateHash \in HashType$
$\langle 2 \rangle 1.$ $\land\ LL1RAM.publicState \in PublicStateType$
$\land\ LL1RAM.privateStateEnc \in PrivateStateEncType$
BY $\langle 1 \rangle 2$, $LL1SubtypeImplicationLemma$DEF $LL1SubtypeImplication$
$\langle 2 \rangle 2.$ $\land\ LL1RAM.publicState \in HashDomain$
$\land\ LL1RAM.privateStateEnc \in HashDomain$
BY $\langle 2 \rangle 1$ DEF $HashDomain$
$\langle 2 \rangle 3.$ QED
BY $\langle 2 \rangle 2$, $HashTypeSafe$DEF $stateHash$
$\langle 1 \rangle 4.$ $historyStateBinding \in HashType$
$\langle 2 \rangle 1.$ $LL1RAM.historySummary \in HashDomain$
$\langle 3 \rangle 1.$ $LL1RAM.historySummary \in HashType$
BY $\langle 1 \rangle 2$, $LL1SubtypeImplicationLemma$DEF $LL1SubtypeImplication$
$\langle 3 \rangle 2.$ QED
BY $\langle 3 \rangle 1$ DEF $HashDomain$
$\langle 2 \rangle 2.$ $stateHash \in HashDomain$

BY $\langle 1 \rangle 3$ DEF *HashDomain*

$\langle 2 \rangle 3$. QED

BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, *HashTypeSafe*DEF *historyStateBinding*

$\langle 1 \rangle 5$. $privateState \in PrivateStateType$

$\langle 2 \rangle 1$. $\land\ LL1NVRAM.symmetricKey \in SymmetricKeyType$

$\land\ LL1RAM.privateStateEnc \in PrivateStateEncType$

BY $\langle 1 \rangle 2$, *LL1SubtypeImplicationLemma*DEF *LL1SubtypeImplication*

$\langle 2 \rangle 2$. QED

BY $\langle 2 \rangle 1$, *SymmetricDecryptionTypeSafe*DEF *privateState*

$\langle 1 \rangle 6$. $sResult \in ServiceResultType$

$\langle 2 \rangle 1$. $LL1RAM.publicState \in PublicStateType$

BY $\langle 1 \rangle 2$, *LL1SubtypeImplicationLemma*DEF *LL1SubtypeImplication*

$\langle 2 \rangle 2$. $privateState \in PrivateStateType$

BY $\langle 1 \rangle 5$

$\langle 2 \rangle 3$. $input \in InputType$

$\langle 3 \rangle 1$. $LL1AvailableInputs \subseteq InputType$

BY $\langle 1 \rangle 2$ DEF *LL1TypeInvariant*

$\langle 3 \rangle 2$. QED

BY $\langle 1 \rangle 1$, $\langle 3 \rangle 1$

$\langle 2 \rangle 4$. QED

BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, $\langle 2 \rangle 3$, *ServiceTypeSafe*DEF *sResult*

$\langle 1 \rangle 7$. $\land\ sResult.newPublicState \in PublicStateType$

$\land\ sResult.newPrivateState \in PrivateStateType$

$\land\ sResult.output \in OutputType$

BY $\langle 1 \rangle 6$ DEF *ServiceResultType*

$\langle 1 \rangle 8$. $newPrivateStateEnc \in PrivateStateEncType$

$\langle 2 \rangle 1$. $LL1NVRAM.symmetricKey \in SymmetricKeyType$

BY $\langle 1 \rangle 2$, *LL1SubtypeImplicationLemma*DEF *LL1SubtypeImplication*

$\langle 2 \rangle 2$. $sResult.newPrivateState \in PrivateStateType$

BY $\langle 1 \rangle 7$

$\langle 2 \rangle 3$. QED

BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, *SymmetricEncryptionTypeSafe*DEF *newPrivateStateEnc*

$\langle 1 \rangle 9$. $newStateHash \in HashType$

$\langle 2 \rangle 1$. $sResult.newPublicState \in HashDomain$

$\langle 3 \rangle 1$. $sResult.newPublicState \in PublicStateType$

BY $\langle 1 \rangle 7$

$\langle 3 \rangle 2$. QED

BY $\langle 3 \rangle 1$ DEF *HashDomain*

$\langle 2 \rangle 2$. $newPrivateStateEnc \in HashDomain$

BY $\langle 1 \rangle 8$ DEF *HashDomain*

$\langle 2 \rangle 3$. QED

BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, *HashTypeSafe*DEF *newStateHash*

$\langle 1 \rangle 10$. $newHistoryStateBinding \in HashType$

$\langle 2 \rangle 1$. $LL1NVRAM.historySummary \in HashDomain$

$\langle 3 \rangle 1$. $LL1NVRAM.historySummary \in HashType$

BY $\langle 1 \rangle 2$, *LL1SubtypeImplicationLemma*DEF *LL1SubtypeImplication*

$\langle 3 \rangle 2$. QED

BY $\langle 3 \rangle 1$ DEF *HashDomain*

$\langle 2 \rangle 2$. $newStateHash \in HashDomain$

BY $\langle 1 \rangle 9$ DEF *HashDomain*

$\langle 2 \rangle 3$. QED

BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, *HashTypeSafe*DEF *newHistoryStateBinding*

$\langle 1 \rangle 11.$ *newAuthenticator* $\in$ *MACType*

$\langle 2 \rangle 1.$ *LL1NVRAM.symmetricKey* $\in$ *SymmetricKeyType*

BY $\langle 1 \rangle 2$, *LL1SubtypeImplicationLemma*DEF *LL1SubtypeImplication*

$\langle 2 \rangle 2.$ *newHistoryStateBinding* $\in$ *HashType*

BY $\langle 1 \rangle 10$

$\langle 2 \rangle 3.$ QED

BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, *GenerateMACTypeSafe*DEF *newAuthenticator*

$\langle 1 \rangle 12.$ QED

BY $\langle 1 \rangle 3$, $\langle 1 \rangle 4$, $\langle 1 \rangle 5$, $\langle 1 \rangle 6$, $\langle 1 \rangle 7$, $\langle 1 \rangle 8$, $\langle 1 \rangle 9$, $\langle 1 \rangle 10$, $\langle 1 \rangle 11$

DEF *stateHash*, *historyStateBinding*, *privateState*, *sResult*, *newPrivateStateEnc*, *newStateHash*, *newHistoryStateBinding*, *newAuthenticator*

The *InclusionInvariantDefsTypeSafeLemma* proves that the definitions within the LET of the *InclusionInvariant* all have the appropriate type. This is a trivial proof that merely walks through the definitions.

THEOREM *InclusionInvariantDefsTypeSafeLemma* $\triangleq$

$\forall$ *input* $\in$ *InputType*,
historySummary $\in$ *HashType*,
publicState $\in$ *PublicStateType*,
privateStateEnc $\in$ *PrivateStateEncType* :
LL1TypeInvariant $\Rightarrow$

LET

stateHash $\triangleq$ *Hash*(*publicState*, *privateStateEnc*)
historyStateBinding $\triangleq$ *Hash*(*historySummary*, *stateHash*)
privateState $\triangleq$ *SymmetricDecrypt*(*LL1NVRAM.symmetricKey*, *privateStateEnc*)
sResult $\triangleq$ *Service*(*publicState*, *privateState*, *input*)
newPrivateStateEnc $\triangleq$
SymmetricEncrypt(*LL1NVRAM.symmetricKey*, *sResult.newPrivateState*)
newStateHash $\triangleq$ *Hash*(*sResult.newPublicState*, *newPrivateStateEnc*)
newHistoryStateBinding $\triangleq$ *Hash*(*LL1NVRAM.historySummary*, *newStateHash*)

IN

$\wedge$ *stateHash* $\in$ *HashType*
$\wedge$ *historyStateBinding* $\in$ *HashType*
$\wedge$ *privateState* $\in$ *PrivateStateType*
$\wedge$ *sResult* $\in$ *ServiceResultType*
$\wedge$ *sResult.newPublicState* $\in$ *PublicStateType*
$\wedge$ *sResult.newPrivateState* $\in$ *PrivateStateType*
$\wedge$ *sResult.output* $\in$ *OutputType*
$\wedge$ *newPrivateStateEnc* $\in$ *PrivateStateEncType*
$\wedge$ *newStateHash* $\in$ *HashType*
$\wedge$ *newHistoryStateBinding* $\in$ *HashType*

$\langle 1 \rangle 1.$ TAKE *input* $\in$ *InputType*,
historySummary $\in$ *HashType*,
publicState $\in$ *PublicStateType*,
privateStateEnc $\in$ *PrivateStateEncType*

$\langle 1 \rangle$ *stateHash* $\triangleq$ *Hash*(*publicState*, *privateStateEnc*)
$\langle 1 \rangle$ *historyStateBinding* $\triangleq$ *Hash*(*historySummary*, *stateHash*)
$\langle 1 \rangle$ *privateState* $\triangleq$ *SymmetricDecrypt*(*LL1NVRAM.symmetricKey*, *privateStateEnc*)
$\langle 1 \rangle$ *sResult* $\triangleq$ *Service*(*publicState*, *privateState*, *input*)
$\langle 1 \rangle$ *newPrivateStateEnc* $\triangleq$
SymmetricEncrypt(*LL1NVRAM.symmetricKey*, *sResult.newPrivateState*)

$\langle 1 \rangle$ $newStateHash \triangleq Hash(sResult.newPublicState, newPrivateStateEnc)$
$\langle 1 \rangle$ $newHistoryStateBinding \triangleq Hash(LL1NVRAM.historySummary, newStateHash)$
$\langle 1 \rangle$ HIDE DEF $stateHash$, $historyStateBinding$, $privateState$, $sResult$,
$newPrivateStateEnc$, $newStateHash$, $newHistoryStateBinding$
$\langle 1 \rangle 2.$ HAVE $LL1TypeInvariant$
$\langle 1 \rangle 3.$ $stateHash \in HashType$
$\langle 2 \rangle 1.$ $\land$ $publicState \in PublicStateType$
$\land$ $privateStateEnc \in PrivateStateEncType$
BY $\langle 1 \rangle 1$
$\langle 2 \rangle 2.$ $\land$ $publicState \in HashDomain$
$\land$ $privateStateEnc \in HashDomain$
BY $\langle 2 \rangle 1$ DEF $HashDomain$
$\langle 2 \rangle 3.$ QED
BY $\langle 2 \rangle 2$, $HashTypeSafe$DEF $stateHash$
$\langle 1 \rangle 4.$ $historyStateBinding \in HashType$
$\langle 2 \rangle 1.$ $historySummary \in HashDomain$
$\langle 3 \rangle 1.$ $historySummary \in HashType$
BY $\langle 1 \rangle 1$
$\langle 3 \rangle 2.$ QED
BY $\langle 3 \rangle 1$ DEF $HashDomain$
$\langle 2 \rangle 2.$ $stateHash \in HashDomain$
BY $\langle 1 \rangle 3$ DEF $HashDomain$
$\langle 2 \rangle 3.$ QED
BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, $HashTypeSafe$DEF $historyStateBinding$
$\langle 1 \rangle 5.$ $privateState \in PrivateStateType$
$\langle 2 \rangle 1.$ $LL1NVRAM.symmetricKey \in SymmetricKeyType$
BY $\langle 1 \rangle 2$, $LL1SubtypeImplicationLemma$DEF $LL1SubtypeImplication$
$\langle 2 \rangle 2.$ $privateStateEnc \in PrivateStateEncType$
BY $\langle 1 \rangle 1$
$\langle 2 \rangle 3.$ QED
BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, $SymmetricDecryptionTypeSafe$DEF $privateState$
$\langle 1 \rangle 6.$ $sResult \in ServiceResultType$
$\langle 2 \rangle 1.$ $publicState \in PublicStateType$
BY $\langle 1 \rangle 1$
$\langle 2 \rangle 2.$ $privateState \in PrivateStateType$
BY $\langle 1 \rangle 5$
$\langle 2 \rangle 3.$ $input \in InputType$
$\langle 3 \rangle 1.$ $LL1AvailableInputs \subseteq InputType$
BY $\langle 1 \rangle 2$ DEF $LL1TypeInvariant$
$\langle 3 \rangle 2.$ QED
BY $\langle 1 \rangle 1$, $\langle 3 \rangle 1$
$\langle 2 \rangle 4.$ QED
BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, $\langle 2 \rangle 3$, $ServiceTypeSafe$DEF $sResult$
$\langle 1 \rangle 7.$ $\land$ $sResult.newPublicState \in PublicStateType$
$\land$ $sResult.newPrivateState \in PrivateStateType$
$\land$ $sResult.output \in OutputType$
BY $\langle 1 \rangle 6$ DEF $ServiceResultType$
$\langle 1 \rangle 8.$ $newPrivateStateEnc \in PrivateStateEncType$
$\langle 2 \rangle 1.$ $LL1NVRAM.symmetricKey \in SymmetricKeyType$
BY $\langle 1 \rangle 2$, $LL1SubtypeImplicationLemma$DEF $LL1SubtypeImplication$
$\langle 2 \rangle 2.$ $sResult.newPrivateState \in PrivateStateType$
BY $\langle 1 \rangle 7$

⟨2⟩3. QED

BY ⟨2⟩1, ⟨2⟩2, *SymmetricEncryptionTypeSafe* DEF *newPrivateStateEnc*

⟨1⟩9. *newStateHash* ∈ *HashType*

⟨2⟩1. *sResult.newPublicState* ∈ *HashDomain*

⟨3⟩1. *sResult.newPublicState* ∈ *PublicStateType*

BY ⟨1⟩7

⟨3⟩2. QED

BY ⟨3⟩1 DEF *HashDomain*

⟨2⟩2. *newPrivateStateEnc* ∈ *HashDomain*

BY ⟨1⟩8 DEF *HashDomain*

⟨2⟩3. QED

BY ⟨2⟩1, ⟨2⟩2, *HashTypeSafe* DEF *newStateHash*

⟨1⟩10. *newHistoryStateBinding* ∈ *HashType*

⟨2⟩1. *LL1NVRAM.historySummary* ∈ *HashDomain*

⟨3⟩1. *LL1NVRAM.historySummary* ∈ *HashType*

BY ⟨1⟩2, *LL1SubtypeImplicationLemma* DEF *LL1SubtypeImplication*

⟨3⟩2. QED

BY ⟨3⟩1 DEF *HashDomain*

⟨2⟩2. *newStateHash* ∈ *HashDomain*

BY ⟨1⟩9 DEF *HashDomain*

⟨2⟩3. QED

BY ⟨2⟩1, ⟨2⟩2, *HashTypeSafe* DEF *newHistoryStateBinding*

⟨1⟩11. QED

BY ⟨1⟩3, ⟨1⟩4, ⟨1⟩5, ⟨1⟩6, ⟨1⟩7, ⟨1⟩8, ⟨1⟩9, ⟨1⟩10

DEF *stateHash*, *historyStateBinding*, *privateState*, *sResult*, *newPrivateStateEnc*, *newStateHash*, *newHistoryStateBinding*

The *CardinalityInvariantDefsTypeSafeLemma* proves that the definition within the LET of the *CardinalityInvariant* has the appropriate type. This is a trivial proof that merely walks through the definitions.

THEOREM *CardinalityInvariantDefsTypeSafeLemma* ≜

∀ *historySummary* ∈ *HashType*, *stateHash* ∈ *HashType* :

LET

historyStateBinding ≜ *Hash*(*historySummary*, *stateHash*)

IN

historyStateBinding ∈ *HashType*

⟨1⟩1. TAKE *historySummary* ∈ *HashType*, *stateHash* ∈ *HashType*

⟨1⟩ *historyStateBinding* ≜ *Hash*(*historySummary*, *stateHash*)

⟨1⟩ HIDE DEF *historyStateBinding*

⟨1⟩2. QED

⟨2⟩1. *historySummary* ∈ *HashDomain*

⟨3⟩1. *historySummary* ∈ *HashType*

BY ⟨1⟩1

⟨3⟩2. QED

BY ⟨3⟩1 DEF *HashDomain*

⟨2⟩2. *stateHash* ∈ *HashDomain*

⟨3⟩1. *stateHash* ∈ *HashType*

BY ⟨1⟩1

⟨3⟩2. QED

BY ⟨3⟩1 DEF *HashDomain*

⟨2⟩3. QED

BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, *HashTypeSafe* DEF *historyStateBinding*

The *UniquenessInvariantDefsTypeSafeLemma* proves that the definitions within the LET of the *UniquenessInvariant* all have the appropriate type. This is a trivial proof that merely walks through the definitions.

THEOREM *UniquenessInvariantDefsTypeSafeLemma* $\triangleq$
$\quad \forall$ *stateHash*1, *stateHash*2 $\in$ *HashType* :
$\quad\quad$ *LL*1*TypeInvariant* $\Rightarrow$
$\quad\quad\quad$ LET
$\quad\quad\quad\quad$ *historyStateBinding*1 $\triangleq$ *Hash*(*LL*1*NVRAM*.*historySummary*, *stateHash*1)
$\quad\quad\quad\quad$ *historyStateBinding*2 $\triangleq$ *Hash*(*LL*1*NVRAM*.*historySummary*, *stateHash*2)
$\quad\quad\quad$ IN
$\quad\quad\quad\quad$ $\land$ *historyStateBinding*1 $\in$ *HashType*
$\quad\quad\quad\quad$ $\land$ *historyStateBinding*2 $\in$ *HashType*

$\langle 1 \rangle 1$. TAKE *stateHash*1, *stateHash*2 $\in$ *HashType*
$\langle 1 \rangle$ *historyStateBinding*1 $\triangleq$ *Hash*(*LL*1*NVRAM*.*historySummary*, *stateHash*1)
$\langle 1 \rangle$ *historyStateBinding*2 $\triangleq$ *Hash*(*LL*1*NVRAM*.*historySummary*, *stateHash*2)
$\langle 1 \rangle$ HIDE DEF *historyStateBinding*1, *historyStateBinding*2
$\langle 1 \rangle 2$. HAVE *LL*1*TypeInvariant*
$\langle 1 \rangle 3$. *LL*1*NVRAM*.*historySummary* $\in$ *HashDomain*
$\quad \langle 2 \rangle 1$. *LL*1*NVRAM*.*historySummary* $\in$ *HashType*
$\quad\quad \langle 3 \rangle 1$. *LL*1*TypeInvariant*
$\quad\quad\quad$ BY $\langle 1 \rangle 2$
$\quad\quad \langle 3 \rangle 2$. QED
$\quad\quad\quad$ BY $\langle 3 \rangle 1$, *LL*1*SubtypeImplicationLemma* DEF *LL*1*SubtypeImplication*
$\quad \langle 2 \rangle 2$. QED
$\quad\quad$ BY $\langle 2 \rangle 1$ DEF *HashDomain*
$\langle 1 \rangle 4$. *historyStateBinding*1 $\in$ *HashType*
$\quad \langle 2 \rangle 1$. *stateHash*1 $\in$ *HashDomain*
$\quad\quad \langle 3 \rangle 1$. *stateHash*1 $\in$ *HashType*
$\quad\quad\quad$ BY $\langle 1 \rangle 1$
$\quad\quad \langle 3 \rangle 2$. QED
$\quad\quad\quad$ BY $\langle 3 \rangle 1$ DEF *HashDomain*
$\quad \langle 2 \rangle 2$. QED
$\quad\quad$ BY $\langle 1 \rangle 3$, $\langle 2 \rangle 1$, *HashTypeSafe* DEF *historyStateBinding*1
$\langle 1 \rangle 5$. *historyStateBinding*2 $\in$ *HashType*
$\quad \langle 2 \rangle 1$. *stateHash*2 $\in$ *HashDomain*
$\quad\quad \langle 3 \rangle 1$. *stateHash*2 $\in$ *HashType*
$\quad\quad\quad$ BY $\langle 1 \rangle 1$
$\quad\quad \langle 3 \rangle 2$. QED
$\quad\quad\quad$ BY $\langle 3 \rangle 1$ DEF *HashDomain*
$\quad \langle 2 \rangle 2$. QED
$\quad\quad$ BY $\langle 1 \rangle 3$, $\langle 2 \rangle 1$, *HashTypeSafe* DEF *historyStateBinding*2

$\langle 1 \rangle 6$. QED
$\quad$ BY $\langle 1 \rangle 4$, $\langle 1 \rangle 5$
$\quad$ DEF *historyStateBinding*1, *historyStateBinding*2

The *LL*1*RefinementDefsTypeSafeLemma* proves that the definitions within the LET of the *LL*1*Refinement* definition all have the appropriate type in the unprimed state. This is a trivial proof that merely walks through the definitions.

THEOREM $LL1RefinementDefsTypeSafeLemma \triangleq$
$LL1Refinement \land LL1TypeInvariant \Rightarrow$
LET
$refPrivateStateEnc \triangleq SymmetricEncrypt(LL1NVRAM.symmetricKey, HLPrivateState)$
$refStateHash \triangleq Hash(HLPublicState, refPrivateStateEnc)$
$refHistoryStateBinding \triangleq Hash(LL1NVRAM.historySummary, refStateHash)$
IN
$\land\ refPrivateStateEnc \in PrivateStateEncType$
$\land\ refStateHash \in HashType$
$\land\ refHistoryStateBinding \in HashType$
$\langle 1 \rangle$ $refPrivateStateEnc \triangleq SymmetricEncrypt(LL1NVRAM.symmetricKey, HLPrivateState)$
$\langle 1 \rangle$ $refStateHash \triangleq Hash(HLPublicState, refPrivateStateEnc)$
$\langle 1 \rangle$ $refHistoryStateBinding \triangleq Hash(LL1NVRAM.historySummary, refStateHash)$
$\langle 1 \rangle$ HIDE DEF $refPrivateStateEnc$, $refStateHash$
$\langle 1 \rangle 1.$ HAVE $LL1Refinement \land LL1TypeInvariant$
$\langle 1 \rangle 2.$ $refPrivateStateEnc \in PrivateStateEncType$
$\langle 2 \rangle 1.$ $LL1NVRAM.symmetricKey \in SymmetricKeyType$
BY $\langle 1 \rangle 1$, $LL1SubtypeImplicationLemma$DEF $LL1SubtypeImplication$
$\langle 2 \rangle 2.$ $HLPrivateState \in PrivateStateType$
BY $\langle 1 \rangle 1$, $ConstantsTypeSafe$DEF $LL1Refinement$
$\langle 2 \rangle 3.$ QED
BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, $SymmetricEncryptionTypeSafe$DEF $refPrivateStateEnc$
$\langle 1 \rangle 3.$ $refStateHash \in HashType$
$\langle 2 \rangle 1.$ $HLPublicState \in HashDomain$
$\langle 3 \rangle 1.$ $HLPublicState \in PublicStateType$
BY $\langle 1 \rangle 1$, $ConstantsTypeSafe$DEF $LL1Refinement$
$\langle 3 \rangle 2.$ QED
BY $\langle 3 \rangle 1$ DEF $HashDomain$
$\langle 2 \rangle 2.$ $refPrivateStateEnc \in HashDomain$
BY $\langle 1 \rangle 2$ DEF $HashDomain$
$\langle 2 \rangle 3.$ QED
BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, $HashTypeSafe$DEF $refStateHash$
$\langle 1 \rangle 4.$ $refHistoryStateBinding \in HashType$
$\langle 2 \rangle 1.$ $LL1NVRAM.historySummary \in HashDomain$
$\langle 3 \rangle 1.$ $LL1NVRAM.historySummary \in HashType$
BY $\langle 1 \rangle 1$, $LL1SubtypeImplicationLemma$DEF $LL1SubtypeImplication$
$\langle 3 \rangle 2.$ QED
BY $\langle 3 \rangle 1$ DEF $HashDomain$
$\langle 2 \rangle 2.$ $refStateHash \in HashDomain$
BY $\langle 1 \rangle 3$ DEF $HashDomain$
$\langle 2 \rangle 3.$ QED
BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, $HashTypeSafe$
$\langle 1 \rangle 5.$ QED
BY $\langle 1 \rangle 2$, $\langle 1 \rangle 3$, $\langle 1 \rangle 4$
DEF $refPrivateStateEnc$, $refStateHash$

The *LL1RefinementPrimeDefsTypeSafeLemma* proves that the definitions within the LET of the *LL1Refinement* definition all have the appropriate type in the primed state. This is a trivial proof that merely walks through the definitions.

THEOREM $LL1RefinementPrimeDefsTypeSafeLemma \triangleq$
$LL1Refinement' \land LL1TypeInvariant' \Rightarrow$

LET

$\quad refPrivateStateEnc \triangleq SymmetricEncrypt(LL1NVRAM.symmetricKey, HLPrivateState)$

$\quad refStateHash \triangleq Hash(HLPublicState, refPrivateStateEnc)$

$\quad refHistoryStateBinding \triangleq Hash(LL1NVRAM.historySummary, refStateHash)$

IN

$\quad \land\ refPrivateStateEnc' \in PrivateStateEncType$

$\quad \land\ refStateHash' \in HashType$

$\quad \land\ refHistoryStateBinding' \in HashType$

$\langle 1 \rangle$ $refPrivateStateEnc \triangleq SymmetricEncrypt(LL1NVRAM.symmetricKey, HLPrivateState)$

$\langle 1 \rangle$ $refStateHash \triangleq Hash(HLPublicState, refPrivateStateEnc)$

$\langle 1 \rangle$ $refHistoryStateBinding \triangleq Hash(LL1NVRAM.historySummary, refStateHash)$

$\langle 1 \rangle$ HIDE DEF $refPrivateStateEnc$, $refStateHash$

$\langle 1 \rangle 1.$ HAVE $LL1Refinement' \land LL1TypeInvariant'$

$\langle 1 \rangle 2.$ $refPrivateStateEnc' \in PrivateStateEncType$

$\langle 2 \rangle 1.$ $LL1NVRAM.symmetricKey' \in SymmetricKeyType$

BY $\langle 1 \rangle 1$, $LL1SubtypeImplicationLemma$DEF $LL1SubtypeImplication$

$\langle 2 \rangle 2.$ $HLPrivateState' \in PrivateStateType$

BY $\langle 1 \rangle 1$, $ConstantsTypeSafe$DEF $LL1Refinement$

$\langle 2 \rangle 3.$ QED

BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, $SymmetricEncryptionTypeSafe$DEF $refPrivateStateEnc$

$\langle 1 \rangle 3.$ $refStateHash' \in HashType$

$\langle 2 \rangle 1.$ $HLPublicState' \in HashDomain$

$\langle 3 \rangle 1.$ $HLPublicState' \in PublicStateType$

BY $\langle 1 \rangle 1$, $ConstantsTypeSafe$DEF $LL1Refinement$

$\langle 3 \rangle 2.$ QED

BY $\langle 3 \rangle 1$ DEF $HashDomain$

$\langle 2 \rangle 2.$ $refPrivateStateEnc' \in HashDomain$

BY $\langle 1 \rangle 2$ DEF $HashDomain$

$\langle 2 \rangle 3.$ QED

BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, $HashTypeSafe$DEF $refStateHash$

$\langle 1 \rangle 4.$ $refHistoryStateBinding' \in HashType$

$\langle 2 \rangle 1.$ $LL1NVRAM.historySummary' \in HashDomain$

$\langle 3 \rangle 1.$ $LL1NVRAM.historySummary' \in HashType$

BY $\langle 1 \rangle 1$, $LL1SubtypeImplicationLemma$DEF $LL1SubtypeImplication$

$\langle 3 \rangle 2.$ QED

BY $\langle 3 \rangle 1$ DEF $HashDomain$

$\langle 2 \rangle 2.$ $refStateHash' \in HashDomain$

BY $\langle 1 \rangle 3$ DEF $HashDomain$

$\langle 2 \rangle 3.$ QED

BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, $HashTypeSafe$

$\langle 1 \rangle 5.$ QED

BY $\langle 1 \rangle 2$, $\langle 1 \rangle 3$, $\langle 1 \rangle 4$

DEF $refPrivateStateEnc$, $refStateHash$

4.3 Proof of Type Safety of the Memoir-Basic Spec

MODULE *MemoirLL1TypeSafety*

This module proves the type safety of the Memoir-Basic spec.

EXTENDS *MemoirLL1TypeLemmas*

THEOREM $LL1TypeSafe \triangleq LL1Spec \Rightarrow \Box LL1TypeInvariant$

The top level of the proof is boilerplate TLA+ for an *Inv*1-style proof. First, we prove that the initial state satisfies *LL1TypeInvariant*. Second, we prove that the *LL1Next* predicate inductively preserves *LL1TypeInvariant*. Third, we use temporal induction to prove that these two conditions satisfy type safety over all behaviors.

⟨1⟩1. $LL1Init \Rightarrow LL1TypeInvariant$

The base case follows directly from the definition of *LL1Init*. There are a bunch of steps, but they are simple expansions of definitions and appeals to the type safety of the initial definitions.

⟨2⟩1. HAVE *LL1Init*

⟨2⟩2. PICK $symmetricKey \in SymmetricKeyType : LL1Init!(symmetricKey)!1$

BY ⟨2⟩1 DEF *LL1Init*

⟨2⟩ $initialPrivateStateEnc \triangleq SymmetricEncrypt(symmetricKey, InitialPrivateState)$

⟨2⟩ $initialStateHash \triangleq Hash(InitialPublicState, initialPrivateStateEnc)$

⟨2⟩ $initialHistoryStateBinding \triangleq Hash(BaseHashValue, initialStateHash)$

⟨2⟩ $initialAuthenticator \triangleq GenerateMAC(symmetricKey, initialHistoryStateBinding)$

⟨2⟩ $initialUntrustedStorage \triangleq [$
$publicState \mapsto InitialPublicState,$
$privateStateEnc \mapsto initialPrivateStateEnc,$
$historySummary \mapsto BaseHashValue,$
$authenticator \mapsto initialAuthenticator]$

⟨2⟩ $initialTrustedStorage \triangleq [$
$historySummary \mapsto BaseHashValue,$
$symmetricKey \mapsto symmetricKey]$

⟨2⟩3. $\land\ initialPrivateStateEnc \in PrivateStateEncType$
$\land\ initialStateHash \in HashType$
$\land\ initialHistoryStateBinding \in HashType$
$\land\ initialAuthenticator \in MACType$
$\land\ initialUntrustedStorage \in LL1UntrustedStorageType$
$\land\ initialTrustedStorage \in LL1TrustedStorageType$

⟨3⟩1. $symmetricKey \in SymmetricKeyType$

BY ⟨2⟩2

⟨3⟩2. QED

BY ⟨3⟩1, *LL1InitDefsTypeSafeLemma*

⟨2⟩ HIDE DEF *initialPrivateStateEnc*, *initialStateHash*, *initialAuthenticator*, *initialUntrustedStorage*, *initialTrustedStorage*

⟨2⟩4. $LL1AvailableInputs \subseteq InputType$

⟨3⟩1. $LL1AvailableInputs = InitialAvailableInputs$

BY ⟨2⟩2

⟨3⟩2. $InitialAvailableInputs \subseteq InputType$

BY *ConstantsTypeSafe*DEF *ConstantsTypeSafe*

⟨3⟩3. QED

BY ⟨3⟩1, ⟨3⟩2

⟨2⟩5. $LL1ObservedOutputs \subseteq OutputType$

⟨3⟩1. $LL1ObservedOutputs = \{\}$

BY ⟨2⟩2

⟨3⟩2. QED

BY ⟨3⟩1

⟨2⟩6. $LL1ObservedAuthenticators \subseteq MACType$
  ⟨3⟩1. $LL1ObservedAuthenticators = \{initialAuthenticator\}$
    BY ⟨2⟩2
    DEF $initialAuthenticator$, $initialHistoryStateBinding$,
      $initialStateHash$, $initialPrivateStateEnc$
  ⟨3⟩2. $initialAuthenticator \in MACType$
    BY ⟨2⟩3
  ⟨3⟩3. QED
    BY ⟨3⟩1, ⟨3⟩2
⟨2⟩7. $LL1Disk \in LL1UntrustedStorageType$
  ⟨3⟩1. $LL1Disk = initialUntrustedStorage$
    BY ⟨2⟩2
    DEF $initialUntrustedStorage$, $initialAuthenticator$,
      $initialHistoryStateBinding$, $initialStateHash$, $initialPrivateStateEnc$
  ⟨3⟩2. $initialUntrustedStorage \in LL1UntrustedStorageType$
    BY ⟨2⟩3
  ⟨3⟩3. QED
    BY ⟨3⟩1, ⟨3⟩2
⟨2⟩8. $LL1RAM \in LL1UntrustedStorageType$
  ⟨3⟩1. $LL1RAM = initialUntrustedStorage$
    BY ⟨2⟩2
    DEF $initialUntrustedStorage$, $initialAuthenticator$,
      $initialHistoryStateBinding$, $initialStateHash$, $initialPrivateStateEnc$
  ⟨3⟩2. $initialUntrustedStorage \in LL1UntrustedStorageType$
    BY ⟨2⟩3
  ⟨3⟩3. QED
    BY ⟨3⟩1, ⟨3⟩2
⟨2⟩9. $LL1NVRAM \in LL1TrustedStorageType$
  ⟨3⟩1. $LL1NVRAM = initialTrustedStorage$
    BY ⟨2⟩2 DEF $initialTrustedStorage$
  ⟨3⟩2. $initialTrustedStorage \in LL1TrustedStorageType$
    BY ⟨2⟩3
  ⟨3⟩3. QED
    BY ⟨3⟩1, ⟨3⟩2
⟨2⟩10. QED
  BY ⟨2⟩4, ⟨2⟩5, ⟨2⟩6, ⟨2⟩7, ⟨2⟩8, ⟨2⟩9 DEF $LL1TypeInvariant$
⟨1⟩2. $LL1TypeInvariant \wedge [LL1Next]_{LL1Vars} \Rightarrow LL1TypeInvariant'$

The induction step is also straightforward. We assume the antecedents of the implication, then show that the consequent holds for all eight $LL1Next$ actions plus stuttering.

⟨2⟩1. HAVE $LL1TypeInvariant \wedge [LL1Next]_{LL1Vars}$
⟨2⟩2. CASE UNCHANGED $LL1Vars$

Type safety is inductively trivial for a stuttering step.

  ⟨3⟩1. $LL1AvailableInputs' \subseteq InputType$
    ⟨4⟩1. $LL1AvailableInputs \subseteq InputType$
      BY ⟨2⟩1 DEF $LL1TypeInvariant$
    ⟨4⟩2. UNCHANGED $LL1AvailableInputs$
      BY ⟨2⟩2 DEF $LL1Vars$
    ⟨4⟩3. QED
      BY ⟨4⟩1, ⟨4⟩2
  ⟨3⟩2. $LL1ObservedOutputs' \subseteq OutputType$
    ⟨4⟩1. $LL1ObservedOutputs \subseteq OutputType$
      BY ⟨2⟩1 DEF $LL1TypeInvariant$

$\langle 4\rangle 2.$ UNCHANGED $LL1ObservedOutputs$
BY $\langle 2\rangle 2$ DEF $LL1Vars$
$\langle 4\rangle 3.$ QED
BY $\langle 4\rangle 1, \langle 4\rangle 2$
$\langle 3\rangle 3.$ $LL1ObservedAuthenticators' \subseteq MACType$
$\langle 4\rangle 1.$ $LL1ObservedAuthenticators \subseteq MACType$
BY $\langle 2\rangle 1$ DEF $LL1TypeInvariant$
$\langle 4\rangle 2.$ UNCHANGED $LL1ObservedAuthenticators$
BY $\langle 2\rangle 2$ DEF $LL1Vars$
$\langle 4\rangle 3.$ QED
BY $\langle 4\rangle 1, \langle 4\rangle 2$
$\langle 3\rangle 4.$ $LL1Disk' \in LL1UntrustedStorageType$
$\langle 4\rangle 1.$ $LL1Disk \in LL1UntrustedStorageType$
BY $\langle 2\rangle 1$ DEF $LL1TypeInvariant$
$\langle 4\rangle 2.$ UNCHANGED $LL1Disk$
BY $\langle 2\rangle 2$ DEF $LL1Vars$
$\langle 4\rangle 3.$ QED
BY $\langle 4\rangle 1, \langle 4\rangle 2$
$\langle 3\rangle 5.$ $LL1RAM' \in LL1UntrustedStorageType$
$\langle 4\rangle 1.$ $LL1RAM \in LL1UntrustedStorageType$
BY $\langle 2\rangle 1$ DEF $LL1TypeInvariant$
$\langle 4\rangle 2.$ UNCHANGED $LL1RAM$
BY $\langle 2\rangle 2$ DEF $LL1Vars$
$\langle 4\rangle 3.$ QED
BY $\langle 4\rangle 1, \langle 4\rangle 2$
$\langle 3\rangle 6.$ $LL1NVRAM' \in LL1TrustedStorageType$
$\langle 4\rangle 1.$ $LL1NVRAM \in LL1TrustedStorageType$
BY $\langle 2\rangle 1$ DEF $LL1TypeInvariant$
$\langle 4\rangle 2.$ UNCHANGED $LL1NVRAM$
BY $\langle 2\rangle 2$ DEF $LL1Vars$
$\langle 4\rangle 3.$ QED
BY $\langle 4\rangle 1, \langle 4\rangle 2$
$\langle 3\rangle 7.$ QED
BY $\langle 3\rangle 1, \langle 3\rangle 2, \langle 3\rangle 3, \langle 3\rangle 4, \langle 3\rangle 5, \langle 3\rangle 6$ DEF $LL1TypeInvariant$
$\langle 2\rangle 3.$ CASE $LL1Next$
$\langle 3\rangle 1.$ CASE $LL1MakeInputAvailable$

Type safety is straightforward for a $LL1MakeInputAvailable$ action.

$\langle 4\rangle 1.$ PICK $input \in InputType : LL1MakeInputAvailable!(input)$
BY $\langle 3\rangle 1$ DEF $LL1MakeInputAvailable$
$\langle 4\rangle 2.$ $LL1AvailableInputs' \subseteq InputType$
$\langle 5\rangle 1.$ $LL1AvailableInputs \subseteq InputType$
BY $\langle 2\rangle 1$ DEF $LL1TypeInvariant$
$\langle 5\rangle 2.$ $LL1AvailableInputs' = LL1AvailableInputs \cup \{input\}$
BY $\langle 4\rangle 1$
$\langle 5\rangle 3.$ $input \in InputType$
BY $\langle 4\rangle 1$
$\langle 5\rangle 4.$ QED
BY $\langle 5\rangle 1, \langle 5\rangle 2, \langle 5\rangle 3$
$\langle 4\rangle 3.$ $LL1ObservedOutputs' \subseteq OutputType$
$\langle 5\rangle 1.$ $LL1ObservedOutputs \subseteq OutputType$
BY $\langle 2\rangle 1$ DEF $LL1TypeInvariant$
$\langle 5\rangle 2.$ UNCHANGED $LL1ObservedOutputs$

BY $\langle 4\rangle 1$

$\langle 5\rangle 3.$ QED

BY $\langle 5\rangle 1$, $\langle 5\rangle 2$

$\langle 4\rangle 4.$ $LL1ObservedAuthenticators' \subseteq MACType$

$\langle 5\rangle 1.$ $LL1ObservedAuthenticators \subseteq MACType$

BY $\langle 2\rangle 1$ DEF $LL1TypeInvariant$

$\langle 5\rangle 2.$ UNCHANGED $LL1ObservedAuthenticators$

BY $\langle 4\rangle 1$

$\langle 5\rangle 3.$ QED

BY $\langle 5\rangle 1$, $\langle 5\rangle 2$

$\langle 4\rangle 5.$ $LL1Disk' \in LL1UntrustedStorageType$

$\langle 5\rangle 1.$ $LL1Disk \in LL1UntrustedStorageType$

BY $\langle 2\rangle 1$ DEF $LL1TypeInvariant$

$\langle 5\rangle 2.$ UNCHANGED $LL1Disk$

BY $\langle 4\rangle 1$

$\langle 5\rangle 3.$ QED

BY $\langle 5\rangle 1$, $\langle 5\rangle 2$

$\langle 4\rangle 6.$ $LL1RAM' \in LL1UntrustedStorageType$

$\langle 5\rangle 1.$ $LL1RAM \in LL1UntrustedStorageType$

BY $\langle 2\rangle 1$ DEF $LL1TypeInvariant$

$\langle 5\rangle 2.$ UNCHANGED $LL1RAM$

BY $\langle 4\rangle 1$

$\langle 5\rangle 3.$ QED

BY $\langle 5\rangle 1$, $\langle 5\rangle 2$

$\langle 4\rangle 7.$ $LL1NVRAM' \in LL1TrustedStorageType$

$\langle 5\rangle 1.$ $LL1NVRAM \in LL1TrustedStorageType$

BY $\langle 2\rangle 1$ DEF $LL1TypeInvariant$

$\langle 5\rangle 2.$ UNCHANGED $LL1NVRAM$

BY $\langle 4\rangle 1$

$\langle 5\rangle 3.$ QED

BY $\langle 5\rangle 1$, $\langle 5\rangle 2$

$\langle 4\rangle 8.$ QED

BY $\langle 4\rangle 2$, $\langle 4\rangle 3$, $\langle 4\rangle 4$, $\langle 4\rangle 5$, $\langle 4\rangle 6$, $\langle 4\rangle 7$ DEF $LL1TypeInvariant$

$\langle 3\rangle 2.$ CASE $LL1PerformOperation$

For a *LL1PerformOperation* action, we just walk through the definitions. Type safety follows directly.

$\langle 4\rangle 1.$ PICK $input \in LL1AvailableInputs : LL1PerformOperation!(input)!1$

BY $\langle 3\rangle 2$ DEF $LL1PerformOperation$

$\langle 4\rangle$ $stateHash \triangleq Hash(LL1RAM.publicState, LL1RAM.privateStateEnc)$

$\langle 4\rangle$ $historyStateBinding \triangleq Hash(LL1RAM.historySummary, stateHash)$

$\langle 4\rangle$ $privateState \triangleq SymmetricDecrypt(LL1NVRAM.symmetricKey, LL1RAM.privateStateEnc)$

$\langle 4\rangle$ $sResult \triangleq Service(LL1RAM.publicState, privateState, input)$

$\langle 4\rangle$ $newPrivateStateEnc \triangleq$
$SymmetricEncrypt(LL1NVRAM.symmetricKey, sResult.newPrivateState)$

$\langle 4\rangle$ $newHistorySummary \triangleq Hash(LL1NVRAM.historySummary, input)$

$\langle 4\rangle$ $newStateHash \triangleq Hash(sResult.newPublicState, newPrivateStateEnc)$

$\langle 4\rangle$ $newHistoryStateBinding \triangleq Hash(newHistorySummary, newStateHash)$

$\langle 4\rangle$ $newAuthenticator \triangleq GenerateMAC(LL1NVRAM.symmetricKey, newHistoryStateBinding)$

$\langle 4\rangle 2.$ $\land\ stateHash \in HashType$
$\land\ historyStateBinding \in HashType$
$\land\ privateState \in PrivateStateType$
$\land\ sResult \in ServiceResultType$
$\land\ sResult.newPublicState \in PublicStateType$

$\land$ $sResult.newPrivateState \in PrivateStateType$
$\land$ $sResult.output \in OutputType$
$\land$ $newPrivateStateEnc \in PrivateStateEncType$
$\land$ $newHistorySummary \in HashType$
$\land$ $newStateHash \in HashType$
$\land$ $newHistoryStateBinding \in HashType$
$\land$ $newAuthenticator \in MACType$

$\langle 5\rangle$1. $input \in LL1AvailableInputs$
BY $\langle 4\rangle$1
$\langle 5\rangle$2. $LL1TypeInvariant$
BY $\langle 2\rangle$1
$\langle 5\rangle$3. QED
BY $\langle 5\rangle$1, $\langle 5\rangle$2, $LL1PerformOperationDefsTypeSafeLemma$

$\langle 4\rangle$ HIDE DEF $stateHash$, $historyStateBinding$, $privateState$, $sResult$, $newPrivateStateEnc$, $newHistorySummary$, $newStateHash$, $newHistoryStateBinding$, $newAuthenticator$

$\langle 4\rangle$3. $LL1AvailableInputs' \subseteq InputType$
$\langle 5\rangle$1. $LL1AvailableInputs \subseteq InputType$
BY $\langle 2\rangle$1 DEF $LL1TypeInvariant$
$\langle 5\rangle$2. UNCHANGED $LL1AvailableInputs$
BY $\langle 4\rangle$1
$\langle 5\rangle$3. QED
BY $\langle 5\rangle$1, $\langle 5\rangle$2

$\langle 4\rangle$4. $LL1ObservedOutputs' \subseteq OutputType$
$\langle 5\rangle$1. $LL1ObservedOutputs \subseteq OutputType$
BY $\langle 2\rangle$1 DEF $LL1TypeInvariant$
$\langle 5\rangle$2. $LL1ObservedOutputs' = LL1ObservedOutputs \cup \{sResult.output\}$
BY $\langle 4\rangle$1 DEF $sResult$, $privateState$
$\langle 5\rangle$3. $sResult.output \in OutputType$
BY $\langle 4\rangle$2
$\langle 5\rangle$4. QED
BY $\langle 5\rangle$1, $\langle 5\rangle$2, $\langle 5\rangle$3

$\langle 4\rangle$5. $LL1ObservedAuthenticators' \subseteq MACType$
$\langle 5\rangle$1. $LL1ObservedAuthenticators \subseteq MACType$
BY $\langle 2\rangle$1 DEF $LL1TypeInvariant$
$\langle 5\rangle$2. $LL1ObservedAuthenticators' =$
$LL1ObservedAuthenticators \cup \{newAuthenticator\}$
BY $\langle 4\rangle$1 DEF $newAuthenticator$, $newHistoryStateBinding$, $newStateHash$, $newHistorySummary$, $newPrivateStateEnc$, $sResult$, $privateState$
$\langle 5\rangle$3. $newAuthenticator \in MACType$
BY $\langle 4\rangle$2
$\langle 5\rangle$4. QED
BY $\langle 5\rangle$1, $\langle 5\rangle$2, $\langle 5\rangle$3

$\langle 4\rangle$6. $LL1Disk' \in LL1UntrustedStorageType$
$\langle 5\rangle$1. $LL1Disk \in LL1UntrustedStorageType$
BY $\langle 2\rangle$1 DEF $LL1TypeInvariant$
$\langle 5\rangle$2. UNCHANGED $LL1Disk$
BY $\langle 4\rangle$1
$\langle 5\rangle$3. QED
BY $\langle 5\rangle$1, $\langle 5\rangle$2

$\langle 4\rangle$7. $LL1RAM' \in LL1UntrustedStorageType$
$\langle 5\rangle$1. $LL1RAM' = [publicState \mapsto sResult.newPublicState,$
$privateStateEnc \mapsto newPrivateStateEnc,$

$historySummary \mapsto newHistorySummary,$
$authenticator \mapsto newAuthenticator]$

BY $\langle 4\rangle 1$ DEF $newAuthenticator$, $newHistoryStateBinding$, $newStateHash$, $newHistorySummary$, $newPrivateStateEnc$, $sResult$, $privateState$

$\langle 5\rangle 2.$ $sResult.newPublicState \in PublicStateType$
BY $\langle 4\rangle 2$

$\langle 5\rangle 3.$ $newPrivateStateEnc \in PrivateStateEncType$
BY $\langle 4\rangle 2$

$\langle 5\rangle 4.$ $newHistorySummary \in HashType$
BY $\langle 4\rangle 2$

$\langle 5\rangle 5.$ $newAuthenticator \in MACType$
BY $\langle 4\rangle 2$

$\langle 5\rangle 6.$ QED
BY $\langle 5\rangle 1$, $\langle 5\rangle 2$, $\langle 5\rangle 3$, $\langle 5\rangle 4$, $\langle 5\rangle 5$ DEF $LL1UntrustedStorageType$

$\langle 4\rangle 8.$ $LL1NVRAM' \in LL1TrustedStorageType$

$\langle 5\rangle 1.$ $LL1NVRAM' = [historySummary \mapsto newHistorySummary,$
$symmetricKey \mapsto LL1NVRAM.symmetricKey]$
BY $\langle 4\rangle 1$ DEF $LL1TypeInvariant$, $newHistorySummary$

$\langle 5\rangle 2.$ $newHistorySummary \in HashType$
BY $\langle 4\rangle 2$

$\langle 5\rangle 3.$ $LL1NVRAM.symmetricKey \in SymmetricKeyType$
BY $\langle 2\rangle 1$, $LL1SubtypeImplicationLemma$DEF $LL1SubtypeImplication$

$\langle 5\rangle 4.$ QED
BY $\langle 5\rangle 1$, $\langle 5\rangle 2$, $\langle 5\rangle 3$ DEF $LL1TrustedStorageType$

$\langle 4\rangle 9.$ QED
BY $\langle 4\rangle 3$, $\langle 4\rangle 4$, $\langle 4\rangle 5$, $\langle 4\rangle 6$, $\langle 4\rangle 7$, $\langle 4\rangle 8$ DEF $LL1TypeInvariant$

$\langle 3\rangle 3.$ CASE $LL1RepeatOperation$

For a *LL1RepeatOperation* action, we just walk through the definitions. Type safety follows directly.

$\langle 4\rangle 1.$ PICK $input \in LL1AvailableInputs : LL1RepeatOperation!(input)!1$
BY $\langle 3\rangle 3$ DEF $LL1RepeatOperation$

$\langle 4\rangle$ $stateHash \triangleq Hash(LL1RAM.publicState, LL1RAM.privateStateEnc)$

$\langle 4\rangle$ $historyStateBinding \triangleq Hash(LL1RAM.historySummary, stateHash)$

$\langle 4\rangle$ $privateState \triangleq SymmetricDecrypt(LL1NVRAM.symmetricKey, LL1RAM.privateStateEnc)$

$\langle 4\rangle$ $sResult \triangleq Service(LL1RAM.publicState, privateState, input)$

$\langle 4\rangle$ $newPrivateStateEnc \triangleq$
$SymmetricEncrypt(LL1NVRAM.symmetricKey, sResult.newPrivateState)$

$\langle 4\rangle$ $newStateHash \triangleq Hash(sResult.newPublicState, newPrivateStateEnc)$

$\langle 4\rangle$ $newHistoryStateBinding \triangleq Hash(LL1NVRAM.historySummary, newStateHash)$

$\langle 4\rangle$ $newAuthenticator \triangleq GenerateMAC(LL1NVRAM.symmetricKey, newHistoryStateBinding)$

$\langle 4\rangle 2.$
$\land\ stateHash \in HashType$
$\land\ historyStateBinding \in HashType$
$\land\ privateState \in PrivateStateType$
$\land\ sResult \in ServiceResultType$
$\land\ sResult.newPublicState \in PublicStateType$
$\land\ sResult.newPrivateState \in PrivateStateType$
$\land\ sResult.output \in OutputType$
$\land\ newPrivateStateEnc \in PrivateStateEncType$
$\land\ newStateHash \in HashType$
$\land\ newHistoryStateBinding \in HashType$
$\land\ newAuthenticator \in MACType$

$\langle 5\rangle 1.$ $input \in LL1AvailableInputs$
BY $\langle 4\rangle 1$

$\langle 5 \rangle 2.$ $LL1TypeInvariant$
BY $\langle 2 \rangle 1$
$\langle 5 \rangle 3.$ QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$, $LL1RepeatOperationDefsTypeSafeLemma$
$\langle 4 \rangle$ HIDE DEF $stateHash$, $historyStateBinding$, $privateState$, $sResult$, $newPrivateStateEnc$, $newStateHash$, $newHistoryStateBinding$, $newAuthenticator$
$\langle 4 \rangle 3.$ $LL1AvailableInputs' \subseteq InputType$
$\langle 5 \rangle 1.$ $LL1AvailableInputs \subseteq InputType$
BY $\langle 2 \rangle 1$ DEF $LL1TypeInvariant$
$\langle 5 \rangle 2.$ UNCHANGED $LL1AvailableInputs$
BY $\langle 4 \rangle 1$
$\langle 5 \rangle 3.$ QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$
$\langle 4 \rangle 4.$ $LL1ObservedOutputs' \subseteq OutputType$
$\langle 5 \rangle 1.$ $LL1ObservedOutputs \subseteq OutputType$
BY $\langle 2 \rangle 1$ DEF $LL1TypeInvariant$
$\langle 5 \rangle 2.$ $LL1ObservedOutputs' = LL1ObservedOutputs \cup \{sResult.output\}$
BY $\langle 4 \rangle 1$ DEF $sResult$, $privateState$
$\langle 5 \rangle 3.$ $sResult.output \in OutputType$
BY $\langle 4 \rangle 2$
$\langle 5 \rangle 4.$ QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$, $\langle 5 \rangle 3$
$\langle 4 \rangle 5.$ $LL1ObservedAuthenticators' \subseteq MACType$
$\langle 5 \rangle 1.$ $LL1ObservedAuthenticators \subseteq MACType$
BY $\langle 2 \rangle 1$ DEF $LL1TypeInvariant$
$\langle 5 \rangle 2.$ $LL1ObservedAuthenticators' = LL1ObservedAuthenticators \cup \{newAuthenticator\}$
BY $\langle 4 \rangle 1$ DEF $newAuthenticator$, $newHistoryStateBinding$, $newStateHash$, $newPrivateStateEnc$, $sResult$, $privateState$
$\langle 5 \rangle 3.$ $newAuthenticator \in MACType$
BY $\langle 4 \rangle 2$
$\langle 5 \rangle 4.$ QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$, $\langle 5 \rangle 3$
$\langle 4 \rangle 6.$ $LL1Disk' \in LL1UntrustedStorageType$
$\langle 5 \rangle 1.$ $LL1Disk \in LL1UntrustedStorageType$
BY $\langle 2 \rangle 1$ DEF $LL1TypeInvariant$
$\langle 5 \rangle 2.$ UNCHANGED $LL1Disk$
BY $\langle 4 \rangle 1$
$\langle 5 \rangle 3.$ QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$
$\langle 4 \rangle 7.$ $LL1RAM' \in LL1UntrustedStorageType$
$\langle 5 \rangle 1.$ $LL1RAM' = [publicState \mapsto sResult.newPublicState, privateStateEnc \mapsto newPrivateStateEnc, historySummary \mapsto LL1NVRAM.historySummary, authenticator \mapsto newAuthenticator]$
BY $\langle 4 \rangle 1$ DEF $newAuthenticator$, $newHistoryStateBinding$, $newStateHash$, $newPrivateStateEnc$, $sResult$, $privateState$
$\langle 5 \rangle 2.$ $sResult.newPublicState \in PublicStateType$
BY $\langle 4 \rangle 2$
$\langle 5 \rangle 3.$ $newPrivateStateEnc \in PrivateStateEncType$
BY $\langle 4 \rangle 2$
$\langle 5 \rangle 4.$ $LL1NVRAM.historySummary \in HashType$

BY $\langle 2 \rangle 1$, $LL1SubtypeImplicationLemma$ DEF $LL1SubtypeImplication$
$\langle 5 \rangle 5$. $newAuthenticator \in MACType$
BY $\langle 4 \rangle 2$
$\langle 5 \rangle 6$. QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$, $\langle 5 \rangle 3$, $\langle 5 \rangle 4$, $\langle 5 \rangle 5$ DEF $LL1UntrustedStorageType$
$\langle 4 \rangle 8$. $LL1NVRAM' \in LL1TrustedStorageType$
$\langle 5 \rangle 1$. $LL1NVRAM \in LL1TrustedStorageType$
BY $\langle 2 \rangle 1$ DEF $LL1TypeInvariant$
$\langle 5 \rangle 2$. UNCHANGED $LL1NVRAM$
BY $\langle 4 \rangle 1$
$\langle 5 \rangle 3$. QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$
$\langle 4 \rangle 9$. QED
BY $\langle 4 \rangle 3$, $\langle 4 \rangle 4$, $\langle 4 \rangle 5$, $\langle 4 \rangle 6$, $\langle 4 \rangle 7$, $\langle 4 \rangle 8$ DEF $LL1TypeInvariant$
$\langle 3 \rangle 4$. CASE $LL1Restart$

Type safety is straightforward for a $LL1Restart$ action.

$\langle 4 \rangle 1$. PICK $untrustedStorage \in LL1UntrustedStorageType$,
$randomSymmetricKey \in SymmetricKeyType \setminus \{LL1NVRAM.symmetricKey\}$,
$hash \in HashType$:
$LL1Restart!(untrustedStorage, randomSymmetricKey, hash)$
BY $\langle 3 \rangle 4$ DEF $LL1Restart$
$\langle 4 \rangle 2$. $LL1AvailableInputs' \subseteq InputType$
$\langle 5 \rangle 1$. $LL1AvailableInputs \subseteq InputType$
BY $\langle 2 \rangle 1$ DEF $LL1TypeInvariant$
$\langle 5 \rangle 2$. UNCHANGED $LL1AvailableInputs$
BY $\langle 4 \rangle 1$
$\langle 5 \rangle 3$. QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$
$\langle 4 \rangle 3$. $LL1ObservedOutputs' \subseteq OutputType$
$\langle 5 \rangle 1$. $LL1ObservedOutputs \subseteq OutputType$
BY $\langle 2 \rangle 1$ DEF $LL1TypeInvariant$
$\langle 5 \rangle 2$. UNCHANGED $LL1ObservedOutputs$
BY $\langle 4 \rangle 1$
$\langle 5 \rangle 3$. QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$
$\langle 4 \rangle 4$. $LL1ObservedAuthenticators' \subseteq MACType$
$\langle 5 \rangle 1$. $LL1ObservedAuthenticators \subseteq MACType$
BY $\langle 2 \rangle 1$ DEF $LL1TypeInvariant$
$\langle 5 \rangle 2$. UNCHANGED $LL1ObservedAuthenticators$
BY $\langle 4 \rangle 1$
$\langle 5 \rangle 3$. QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$
$\langle 4 \rangle 5$. $LL1Disk' \in LL1UntrustedStorageType$
$\langle 5 \rangle 1$. $LL1Disk \in LL1UntrustedStorageType$
BY $\langle 2 \rangle 1$ DEF $LL1TypeInvariant$
$\langle 5 \rangle 2$. UNCHANGED $LL1Disk$
BY $\langle 4 \rangle 1$
$\langle 5 \rangle 3$. QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$
$\langle 4 \rangle 6$. $LL1RAM' \in LL1UntrustedStorageType$
$\langle 5 \rangle 1$. $untrustedStorage \in LL1UntrustedStorageType$
BY $\langle 4 \rangle 1$

$\langle 5 \rangle 2.$ $LL1RAM' = untrustedStorage$
BY $\langle 4 \rangle 1$
$\langle 5 \rangle 3.$ QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$
$\langle 4 \rangle 7.$ $LL1NVRAM' \in LL1TrustedStorageType$
$\langle 5 \rangle 1.$ $LL1NVRAM \in LL1TrustedStorageType$
BY $\langle 2 \rangle 1$ DEF $LL1TypeInvariant$
$\langle 5 \rangle 2.$ UNCHANGED $LL1NVRAM$
BY $\langle 4 \rangle 1$
$\langle 5 \rangle 3.$ QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$
$\langle 4 \rangle 8.$ QED
BY $\langle 4 \rangle 2$, $\langle 4 \rangle 3$, $\langle 4 \rangle 4$, $\langle 4 \rangle 5$, $\langle 4 \rangle 6$, $\langle 4 \rangle 7$ DEF $LL1TypeInvariant$
$\langle 3 \rangle 5.$ CASE $LL1ReadDisk$
Type safety is straightforward for a $LL1ReadDisk$ action.
$\langle 4 \rangle 1.$ $LL1AvailableInputs' \subseteq InputType$
$\langle 5 \rangle 1.$ $LL1AvailableInputs \subseteq InputType$
BY $\langle 2 \rangle 1$ DEF $LL1TypeInvariant$
$\langle 5 \rangle 2.$ UNCHANGED $LL1AvailableInputs$
BY $\langle 3 \rangle 5$ DEF $LL1ReadDisk$
$\langle 5 \rangle 3.$ QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$
$\langle 4 \rangle 2.$ $LL1ObservedOutputs' \subseteq OutputType$
$\langle 5 \rangle 1.$ $LL1ObservedOutputs \subseteq OutputType$
BY $\langle 2 \rangle 1$ DEF $LL1TypeInvariant$
$\langle 5 \rangle 2.$ UNCHANGED $LL1ObservedOutputs$
BY $\langle 3 \rangle 5$ DEF $LL1ReadDisk$
$\langle 5 \rangle 3.$ QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$
$\langle 4 \rangle 3.$ $LL1ObservedAuthenticators' \subseteq MACType$
$\langle 5 \rangle 1.$ $LL1ObservedAuthenticators \subseteq MACType$
BY $\langle 2 \rangle 1$ DEF $LL1TypeInvariant$
$\langle 5 \rangle 2.$ UNCHANGED $LL1ObservedAuthenticators$
BY $\langle 3 \rangle 5$ DEF $LL1ReadDisk$
$\langle 5 \rangle 3.$ QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$
$\langle 4 \rangle 4.$ $LL1Disk' \in LL1UntrustedStorageType$
$\langle 5 \rangle 1.$ $LL1Disk \in LL1UntrustedStorageType$
BY $\langle 2 \rangle 1$ DEF $LL1TypeInvariant$
$\langle 5 \rangle 2.$ UNCHANGED $LL1Disk$
BY $\langle 3 \rangle 5$ DEF $LL1ReadDisk$
$\langle 5 \rangle 3.$ QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$
$\langle 4 \rangle 5.$ $LL1RAM' \in LL1UntrustedStorageType$
$\langle 5 \rangle 1.$ $LL1Disk \in LL1UntrustedStorageType$
BY $\langle 2 \rangle 1$ DEF $LL1TypeInvariant$
$\langle 5 \rangle 2.$ $LL1RAM' = LL1Disk$
BY $\langle 3 \rangle 5$ DEF $LL1ReadDisk$
$\langle 5 \rangle 3.$ QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$
$\langle 4 \rangle 6.$ $LL1NVRAM' \in LL1TrustedStorageType$
$\langle 5 \rangle 1.$ $LL1NVRAM \in LL1TrustedStorageType$

BY $\langle 2 \rangle 1$ DEF $LL1TypeInvariant$
$\langle 5 \rangle 2$. UNCHANGED $LL1NVRAM$
BY $\langle 3 \rangle 5$ DEF $LL1ReadDisk$
$\langle 5 \rangle 3$. QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$
$\langle 4 \rangle 7$. QED
BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$, $\langle 4 \rangle 3$, $\langle 4 \rangle 4$, $\langle 4 \rangle 5$, $\langle 4 \rangle 6$ DEF $LL1TypeInvariant$
$\langle 3 \rangle 6$. CASE $LL1WriteDisk$

Type safety is straightforward for a $LL1WriteDisk$ action.

$\langle 4 \rangle 1$. $LL1AvailableInputs' \subseteq InputType$
$\langle 5 \rangle 1$. $LL1AvailableInputs \subseteq InputType$
BY $\langle 2 \rangle 1$ DEF $LL1TypeInvariant$
$\langle 5 \rangle 2$. UNCHANGED $LL1AvailableInputs$
BY $\langle 3 \rangle 6$ DEF $LL1WriteDisk$
$\langle 5 \rangle 3$. QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$
$\langle 4 \rangle 2$. $LL1ObservedOutputs' \subseteq OutputType$
$\langle 5 \rangle 1$. $LL1ObservedOutputs \subseteq OutputType$
BY $\langle 2 \rangle 1$ DEF $LL1TypeInvariant$
$\langle 5 \rangle 2$. UNCHANGED $LL1ObservedOutputs$
BY $\langle 3 \rangle 6$ DEF $LL1WriteDisk$
$\langle 5 \rangle 3$. QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$
$\langle 4 \rangle 3$. $LL1ObservedAuthenticators' \subseteq MACType$
$\langle 5 \rangle 1$. $LL1ObservedAuthenticators \subseteq MACType$
BY $\langle 2 \rangle 1$ DEF $LL1TypeInvariant$
$\langle 5 \rangle 2$. UNCHANGED $LL1ObservedAuthenticators$
BY $\langle 3 \rangle 6$ DEF $LL1WriteDisk$
$\langle 5 \rangle 3$. QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$
$\langle 4 \rangle 4$. $LL1Disk' \in LL1UntrustedStorageType$
$\langle 5 \rangle 1$. $LL1RAM \in LL1UntrustedStorageType$
BY $\langle 2 \rangle 1$ DEF $LL1TypeInvariant$
$\langle 5 \rangle 2$. $LL1Disk' = LL1RAM$
BY $\langle 3 \rangle 6$ DEF $LL1WriteDisk$
$\langle 5 \rangle 3$. QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$
$\langle 4 \rangle 5$. $LL1RAM' \in LL1UntrustedStorageType$
$\langle 5 \rangle 1$. $LL1RAM \in LL1UntrustedStorageType$
BY $\langle 2 \rangle 1$ DEF $LL1TypeInvariant$
$\langle 5 \rangle 2$. UNCHANGED $LL1RAM$
BY $\langle 3 \rangle 6$ DEF $LL1WriteDisk$
$\langle 5 \rangle 3$. QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$
$\langle 4 \rangle 6$. $LL1NVRAM' \in LL1TrustedStorageType$
$\langle 5 \rangle 1$. $LL1NVRAM \in LL1TrustedStorageType$
BY $\langle 2 \rangle 1$ DEF $LL1TypeInvariant$
$\langle 5 \rangle 2$. UNCHANGED $LL1NVRAM$
BY $\langle 3 \rangle 6$ DEF $LL1WriteDisk$
$\langle 5 \rangle 3$. QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$
$\langle 4 \rangle 7$. QED

BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$, $\langle 4 \rangle 3$, $\langle 4 \rangle 4$, $\langle 4 \rangle 5$, $\langle 4 \rangle 6$ DEF $LL1TypeInvariant$

$\langle 3 \rangle 7$. CASE $LL1CorruptRAM$

Type safety is straightforward for a $LL1CorruptRAM$ action.

$\langle 4 \rangle 1$. PICK $untrustedStorage \in LL1UntrustedStorageType$,
$fakeSymmetricKey \in SymmetricKeyType \setminus \{LL1NVRAM.symmetricKey\}$,
$hash \in HashType$:
$LL1CorruptRAM!(untrustedStorage, fakeSymmetricKey, hash)$
BY $\langle 3 \rangle 7$ DEF $LL1CorruptRAM$

$\langle 4 \rangle 2$. $LL1AvailableInputs' \subseteq InputType$
$\langle 5 \rangle 1$. $LL1AvailableInputs \subseteq InputType$
BY $\langle 2 \rangle 1$ DEF $LL1TypeInvariant$
$\langle 5 \rangle 2$. UNCHANGED $LL1AvailableInputs$
BY $\langle 4 \rangle 1$
$\langle 5 \rangle 3$. QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$

$\langle 4 \rangle 3$. $LL1ObservedOutputs' \subseteq OutputType$
$\langle 5 \rangle 1$. $LL1ObservedOutputs \subseteq OutputType$
BY $\langle 2 \rangle 1$ DEF $LL1TypeInvariant$
$\langle 5 \rangle 2$. UNCHANGED $LL1ObservedOutputs$
BY $\langle 4 \rangle 1$
$\langle 5 \rangle 3$. QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$

$\langle 4 \rangle 4$. $LL1ObservedAuthenticators' \subseteq MACType$
$\langle 5 \rangle 1$. $LL1ObservedAuthenticators \subseteq MACType$
BY $\langle 2 \rangle 1$ DEF $LL1TypeInvariant$
$\langle 5 \rangle 2$. UNCHANGED $LL1ObservedAuthenticators$
BY $\langle 4 \rangle 1$
$\langle 5 \rangle 3$. QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$

$\langle 4 \rangle 5$. $LL1Disk' \in LL1UntrustedStorageType$
$\langle 5 \rangle 1$. $LL1Disk \in LL1UntrustedStorageType$
BY $\langle 2 \rangle 1$ DEF $LL1TypeInvariant$
$\langle 5 \rangle 2$. UNCHANGED $LL1Disk$
BY $\langle 4 \rangle 1$
$\langle 5 \rangle 3$. QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$

$\langle 4 \rangle 6$. $LL1RAM' \in LL1UntrustedStorageType$
$\langle 5 \rangle 1$. $untrustedStorage \in LL1UntrustedStorageType$
BY $\langle 4 \rangle 1$
$\langle 5 \rangle 2$. $LL1RAM' = untrustedStorage$
BY $\langle 4 \rangle 1$
$\langle 5 \rangle 3$. QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$

$\langle 4 \rangle 7$. $LL1NVRAM' \in LL1TrustedStorageType$
$\langle 5 \rangle 1$. $LL1NVRAM \in LL1TrustedStorageType$
BY $\langle 2 \rangle 1$ DEF $LL1TypeInvariant$
$\langle 5 \rangle 2$. UNCHANGED $LL1NVRAM$
BY $\langle 4 \rangle 1$
$\langle 5 \rangle 3$. QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$

$\langle 4 \rangle 8$. QED
BY $\langle 4 \rangle 2$, $\langle 4 \rangle 3$, $\langle 4 \rangle 4$, $\langle 4 \rangle 5$, $\langle 4 \rangle 6$, $\langle 4 \rangle 7$ DEF $LL1TypeInvariant$

$\langle 3 \rangle 8.$ CASE $LL1RestrictedCorruption$

Type safety is straightforward for a $LL1RestrictedCorruption$ action.

$\langle 4 \rangle 2.$ $LL1AvailableInputs' \subseteq InputType$
$\langle 5 \rangle 1.$ $LL1AvailableInputs \subseteq InputType$
BY $\langle 2 \rangle 1$ DEF $LL1TypeInvariant$
$\langle 5 \rangle 2.$ UNCHANGED $LL1AvailableInputs$
BY $\langle 3 \rangle 8$ DEF $LL1RestrictedCorruption$
$\langle 5 \rangle 3.$ QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$
$\langle 4 \rangle 3.$ $LL1ObservedOutputs' \subseteq OutputType$
$\langle 5 \rangle 1.$ $LL1ObservedOutputs \subseteq OutputType$
BY $\langle 2 \rangle 1$ DEF $LL1TypeInvariant$
$\langle 5 \rangle 2.$ UNCHANGED $LL1ObservedOutputs$
BY $\langle 3 \rangle 8$ DEF $LL1RestrictedCorruption$
$\langle 5 \rangle 3.$ QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$
$\langle 4 \rangle 4.$ $LL1ObservedAuthenticators' \subseteq MACType$
$\langle 5 \rangle 1.$ $LL1ObservedAuthenticators \subseteq MACType$
BY $\langle 2 \rangle 1$ DEF $LL1TypeInvariant$
$\langle 5 \rangle 2.$ UNCHANGED $LL1ObservedAuthenticators$
BY $\langle 3 \rangle 8$ DEF $LL1RestrictedCorruption$
$\langle 5 \rangle 3.$ QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$
$\langle 4 \rangle 5.$ $LL1Disk' \in LL1UntrustedStorageType$
$\langle 5 \rangle 1.$ $LL1Disk \in LL1UntrustedStorageType$
BY $\langle 2 \rangle 1$ DEF $LL1TypeInvariant$
$\langle 5 \rangle 2.$ UNCHANGED $LL1Disk$
BY $\langle 3 \rangle 8$ DEF $LL1RestrictedCorruption$
$\langle 5 \rangle 3.$ QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$
$\langle 4 \rangle 6.$ $LL1RAM' \in LL1UntrustedStorageType$
$\langle 5 \rangle 1.$ CASE $LL1RestrictedCorruption!ram!unchanged$
$\langle 6 \rangle 1.$ $LL1RAM \in LL1UntrustedStorageType$
BY $\langle 2 \rangle 1$ DEF $LL1TypeInvariant$
$\langle 6 \rangle 2.$ UNCHANGED $LL1RAM$
BY $\langle 5 \rangle 1$
$\langle 6 \rangle 3.$ QED
BY $\langle 6 \rangle 1$, $\langle 6 \rangle 2$
$\langle 5 \rangle 2.$ CASE $LL1RestrictedCorruption!ram!trashed$
$\langle 6 \rangle 1.$ PICK $untrustedStorage \in LL1UntrustedStorageType$,
$randomSymmetricKey \in SymmetricKeyType \setminus \{LL1NVRAM.symmetricKey\}$,
$hash \in HashType$:
$LL1RestrictedCorruption!ram!trashed!($
$untrustedStorage, randomSymmetricKey, hash)$
BY $\langle 5 \rangle 2$
$\langle 6 \rangle 2.$ $untrustedStorage \in LL1UntrustedStorageType$
BY $\langle 6 \rangle 1$
$\langle 6 \rangle 3.$ $LL1RAM' = untrustedStorage$
BY $\langle 6 \rangle 1$
$\langle 6 \rangle 4.$ QED
BY $\langle 6 \rangle 2$, $\langle 6 \rangle 3$
$\langle 5 \rangle 3.$ QED

BY $\langle 3\rangle 8$, $\langle 5\rangle 1$, $\langle 5\rangle 2$ DEF *LL1RestrictedCorruption*

$\langle 4\rangle 7$. $LL1NVRAM' \in LL1TrustedStorageType$

$\langle 5\rangle 1$. PICK $garbageHistorySummary \in HashType :$
$LL1RestrictedCorruption!nvram!(garbageHistorySummary)$

BY $\langle 3\rangle 8$ DEF *LL1RestrictedCorruption*

$\langle 5\rangle 2$. $garbageHistorySummary \in HashType$

BY $\langle 5\rangle 1$

$\langle 5\rangle 3$. $LL1NVRAM.symmetricKey \in SymmetricKeyType$

BY $\langle 2\rangle 1$, *LL1SubtypeImplicationLemma*DEF *LL1SubtypeImplication*

$\langle 5\rangle 4$. $LL1NVRAM' = [historySummary \mapsto garbageHistorySummary,$
$symmetricKey \mapsto LL1NVRAM.symmetricKey]$

BY $\langle 5\rangle 1$

$\langle 5\rangle 5$. QED

BY $\langle 5\rangle 2$, $\langle 5\rangle 3$, $\langle 5\rangle 4$ DEF *LL1TrustedStorageType*

$\langle 4\rangle 8$. QED

BY $\langle 4\rangle 2$, $\langle 4\rangle 3$, $\langle 4\rangle 4$, $\langle 4\rangle 5$, $\langle 4\rangle 6$, $\langle 4\rangle 7$ DEF *LL1TypeInvariant*

$\langle 3\rangle 9$. QED

BY $\langle 2\rangle 3$, $\langle 3\rangle 1$, $\langle 3\rangle 2$, $\langle 3\rangle 3$, $\langle 3\rangle 4$, $\langle 3\rangle 5$, $\langle 3\rangle 6$, $\langle 3\rangle 7$, $\langle 3\rangle 8$ DEF *LL1Next*

$\langle 2\rangle 4$. QED

BY $\langle 2\rangle 1$, $\langle 2\rangle 2$, $\langle 2\rangle 3$

$\langle 1\rangle 3$. QED

Using the *Inv*1 proof rule, the base case and the induction step together imply that the invariant always holds.

$\langle 2\rangle 1$. $LL1TypeInvariant \land \Box[LL1Next]_{LL1Vars} \Rightarrow \Box LL1TypeInvariant$

BY $\langle 1\rangle 2$, *Inv*1

$\langle 2\rangle 2$. QED

BY $\langle 1\rangle 1$, $\langle 2\rangle 1$ DEF *LL1Spec*

4.4 Proofs of Lemmas that Support Memoir-Basic Invariance Proofs

MODULE *MemoirLL1InvarianceLemmas*

This module states and proves several lemmas that are useful for proving the Memoir-Basic invariance properties.

The lemmas in this module are:
SymmetricKeyConstantLemma
LL1NVRAMHistorySummaryUncorruptedUnchangedLemma
LL1RepeatOperationUnchangedObseredOutputsLemma
LL1RepeatOperationUnchangedAuthenticatedHistoryStateBindingsLemma
LL1RAMUnforgeabilityUnchangedLemma
LL1DiskUnforgeabilityUnchangedLemma
InclusionUnchangedLemma
CardinalityUnchangedLemma
UniquenessUnchangedLemma
UnchangedAuthenticatedHistoryStateBindingsLemma

EXTENDS *MemoirLL1SupplementalInvariants*

Proof relating to cardinality require some basic properties of inequalities on natural numbers. The prover requires that we state these explicitly.

THEOREM $LEQTransitive \triangleq \forall n, m, q \in Nat : n \leq m \wedge m \leq q \Rightarrow n \leq q$
  OBVIOUS $\{by(isabelle \text{ “}(auto\ dest : nat\_leq\_trans)\text{”})\}$

THEOREM $GEQorLT \triangleq \forall n, m \in Nat : n \geq m \equiv \neg(m > n)$
  OBVIOUS $\{by(isabelle \text{ “}(auto\ simp : nat\_not\_less\ dest : nat\_leq\_less\_trans)\text{”})\}$

The *SymmetricKeyConstantLemma* states that the *LL1Next* actions do not change the value of the symmetric key in *NVRAM*. The proof follows directly from the definition of the actions.

THEOREM $SymmetricKeyConstantLemma \triangleq$
  $[LL1Next]_{LL1Vars} \Rightarrow$ UNCHANGED $LL1NVRAM.symmetricKey$
 $\langle 1\rangle 1.$ HAVE $[LL1Next]_{LL1Vars}$
 $\langle 1\rangle 2.$ CASE UNCHANGED $LL1Vars$
  $\langle 2\rangle 1.$ UNCHANGED $LL1NVRAM$
   BY $\langle 1\rangle 2$ DEF $LL1Vars$
  $\langle 2\rangle 2.$ QED
   BY $\langle 2\rangle 1$
 $\langle 1\rangle 3.$ CASE $LL1Next$
  $\langle 2\rangle 1.$ CASE $LL1MakeInputAvailable$
   $\langle 3\rangle 1.$ UNCHANGED $LL1NVRAM$
    BY $\langle 2\rangle 1$ DEF $LL1MakeInputAvailable$
   $\langle 3\rangle 2.$ QED
    BY $\langle 3\rangle 1$, $HashCardinalityAccumulative$
  $\langle 2\rangle 2.$ CASE $LL1PerformOperation$
   $\langle 3\rangle 1.$ PICK $input \in LL1AvailableInputs : LL1PerformOperation!(input)!1$
    BY $\langle 2\rangle 2$ DEF $LL1PerformOperation$
   $\langle 3\rangle 2.$ $LL1NVRAM' = [historySummary \mapsto LL1PerformOperation!(input)!newHistorySummary,$
      $symmetricKey \mapsto LL1NVRAM.symmetricKey]$
    BY $\langle 3\rangle 1$
   $\langle 3\rangle 3.$ $LL1NVRAM.symmetricKey' = LL1NVRAM.symmetricKey$
    BY $\langle 3\rangle 2$
   $\langle 3\rangle 4.$ QED

BY $\langle 3\rangle 3$
$\langle 2\rangle 3.$ CASE $LL1RepeatOperation$
$\langle 3\rangle 1.$ UNCHANGED $LL1NVRAM$
BY $\langle 2\rangle 3$ DEF $LL1RepeatOperation$
$\langle 3\rangle 2.$ QED
BY $\langle 3\rangle 1$
$\langle 2\rangle 4.$ CASE $LL1Restart$
$\langle 3\rangle 1.$ UNCHANGED $LL1NVRAM$
BY $\langle 2\rangle 4$ DEF $LL1Restart$
$\langle 3\rangle 2.$ QED
BY $\langle 3\rangle 1$
$\langle 2\rangle 5.$ CASE $LL1ReadDisk$
$\langle 3\rangle 1.$ UNCHANGED $LL1NVRAM$
BY $\langle 2\rangle 5$ DEF $LL1ReadDisk$
$\langle 3\rangle 2.$ QED
BY $\langle 3\rangle 1$
$\langle 2\rangle 6.$ CASE $LL1WriteDisk$
$\langle 3\rangle 1.$ UNCHANGED $LL1NVRAM$
BY $\langle 2\rangle 6$ DEF $LL1WriteDisk$
$\langle 3\rangle 2.$ QED
BY $\langle 3\rangle 1$
$\langle 2\rangle 7.$ CASE $LL1CorruptRAM$
$\langle 3\rangle 1.$ UNCHANGED $LL1NVRAM$
BY $\langle 2\rangle 7$ DEF $LL1CorruptRAM$
$\langle 3\rangle 2.$ QED
BY $\langle 3\rangle 1$
$\langle 2\rangle 8.$ CASE $LL1RestrictedCorruption$
$\langle 3\rangle 1.$ PICK $garbageHistorySummary \in HashType :$
$LL1RestrictedCorruption!nvram!(garbageHistorySummary)$
BY $\langle 2\rangle 8$ DEF $LL1RestrictedCorruption$
$\langle 3\rangle 2.$ $LL1NVRAM' = [historySummary \mapsto garbageHistorySummary,$
$symmetricKey \mapsto LL1NVRAM.symmetricKey]$
BY $\langle 3\rangle 1$
$\langle 3\rangle 3.$ QED
BY $\langle 3\rangle 2$
$\langle 2\rangle 9.$ QED
BY $\langle 1\rangle 3$, $\langle 2\rangle 1$, $\langle 2\rangle 2$, $\langle 2\rangle 3$, $\langle 2\rangle 4$, $\langle 2\rangle 5$, $\langle 2\rangle 6$, $\langle 2\rangle 7$, $\langle 2\rangle 8$ DEF $LL1Next$
$\langle 1\rangle 4.$ QED
BY $\langle 1\rangle 1$, $\langle 1\rangle 2$, $\langle 1\rangle 3$

The $LL1NVRAMHistorySummaryUncorruptedUnchangedLemma$ states that, if there are no changes to the $NVRAM$ or to the authentication status of any history state binding, then the truth value of the $LL1NVRAMHistorySummaryUncorrupted$ predicate is unchanged.

THEOREM $LL1NVRAMHistorySummaryUncorruptedUnchangedLemma \triangleq$
$(\land\ LL1TypeInvariant$
$\land$ UNCHANGED $LL1NVRAM$
$\land\ \forall\, historyStateBinding1 \in HashType :$
UNCHANGED $LL1HistoryStateBindingAuthenticated(historyStateBinding1))$
$\Rightarrow$
UNCHANGED $LL1NVRAMHistorySummaryUncorrupted$

We begin by assuming the antecedent.

$\langle 1\rangle 1.$ HAVE $\land$ *LL1TypeInvariant*
$\land$ UNCHANGED *LL1NVRAM*
$\land$ $\forall$ *historyStateBinding1* $\in$ *HashType* :
UNCHANGED *LL1HistoryStateBindingAuthenticated*(*historyStateBinding1*)

We first consider the case in which the truth value of the predicate is true.

$\langle 1\rangle 2.$ CASE *LL1NVRAMHistorySummaryUncorrupted* = TRUE

To show that the value is unchanged, it suffices to show that the value is true in the primed state.

$\langle 2\rangle 1.$ SUFFICES
ASSUME TRUE
PROVE *LL1NVRAMHistorySummaryUncorrupted*$'$ = TRUE
BY $\langle 1\rangle 2$

We pick some state hash for which the *LL1NVRAMHistorySummaryUncorrupted* is true in the unprimed state.

$\langle 2\rangle 2.$ PICK *stateHash* $\in$ *HashType* :
LL1NVRAMHistorySummaryUncorrupted!(*stateHash*)!1
BY $\langle 1\rangle 2$ DEF *LL1NVRAMHistorySummaryUncorrupted*

We copy the definition from the LET in *LL1NVRAMHistorySummaryUncorrupted*.

$\langle 2\rangle$ *historyStateBinding* $\triangleq$ *Hash*(*LL1NVRAM.historySummary*, *stateHash*)

The *LL1NVRAMHistorySummaryUncorrupted* predicate has two conditions. First, that there exists a state hash in *HashType*, for which we have a witness.

$\langle 2\rangle 3.$ *stateHash* $\in$ *HashType*
BY $\langle 2\rangle 2$

The second condition is that the history state binding is authenticated in the primed state.

$\langle 2\rangle 4.$ *LL1HistoryStateBindingAuthenticated*(*historyStateBinding*)$'$

We will use the assumption that there is no change to the authentication status of any history state binding.

$\langle 3\rangle 1.$ $\forall$ *historyStateBinding1* $\in$ *HashType* :
UNCHANGED *LL1HistoryStateBindingAuthenticated*(*historyStateBinding1*)
BY $\langle 1\rangle 1$

This requires that we prove the type of the history state binding.

$\langle 3\rangle 2.$ *historyStateBinding* $\in$ *HashType*
$\langle 4\rangle 1.$ *LL1NVRAM.historySummary* $\in$ *HashDomain*
$\langle 5\rangle 1.$ *LL1NVRAM.historySummary* $\in$ *HashType*
$\langle 6\rangle 1.$ *LL1TypeInvariant*
BY $\langle 1\rangle 1$
$\langle 6\rangle 2.$ QED
BY $\langle 6\rangle 1$, *LL1SubtypeImplicationLemma*DEF *LL1SubtypeImplication*
$\langle 5\rangle 2.$ QED
BY $\langle 5\rangle 1$ DEF *HashDomain*
$\langle 4\rangle 2.$ *stateHash* $\in$ *HashDomain*
BY $\langle 2\rangle 3$ DEF *HashDomain*
$\langle 4\rangle 3.$ QED
BY $\langle 4\rangle 1$, $\langle 4\rangle 2$, *HashTypeSafe*DEF *historyStateBinding*

We know that the history state binding is authenticated in the unprimed state.

$\langle 3\rangle 3.$ *LL1HistoryStateBindingAuthenticated*(*historyStateBinding*)
BY $\langle 2\rangle 2$

By applying the assumption that there is no change to the authentication status of any history state binding, we show that the history state binding is authenticated in the primed state.

$\langle 3\rangle 4.$ QED
BY $\langle 3\rangle 1$, $\langle 3\rangle 2$, $\langle 3\rangle 3$

Since both conditions are satisfied, the *LL1NVRAMHistorySummaryUncorrupted* predicate is true in the primed state.

$\langle 2 \rangle 5.$ QED

BY $\langle 2 \rangle 3$, $\langle 2 \rangle 4$ DEF *LL1NVRAMHistorySummaryUncorrupted*

We then consider the case in which the truth value of the predicate is false.

$\langle 1 \rangle 3.$ CASE $LL1NVRAMHistorySummaryUncorrupted = \text{FALSE}$

To show that the value is unchanged, it suffices to show that if the value were unequal to false in the primed state, we would have a contradiction.

$\langle 2 \rangle 1.$ SUFFICES

ASSUME $LL1NVRAMHistorySummaryUncorrupted' \neq \text{FALSE}$

PROVE FALSE

BY $\langle 1 \rangle 3$

$\langle 2 \rangle 2.$ $LL1NVRAMHistorySummaryUncorrupted'$

$\langle 3 \rangle 1.$ $LL1NVRAMHistorySummaryUncorrupted' \in \text{BOOLEAN}$

BY DEF *LL1NVRAMHistorySummaryUncorrupted*

$\langle 3 \rangle 2.$ QED

BY $\langle 2 \rangle 1$, $\langle 3 \rangle 1$

We pick some state hash for which the *LL1NVRAMHistorySummaryUncorrupted* is true in the primed state.

$\langle 2 \rangle 3.$ PICK $stateHash \in HashType$:

$LL1NVRAMHistorySummaryUncorrupted!(stateHash)!1'$

BY $\langle 2 \rangle 2$ DEF *LL1NVRAMHistorySummaryUncorrupted*

We copy the definition from the LET in *LL1NVRAMHistorySummaryUncorrupted*.

$\langle 2 \rangle$ $historyStateBinding \triangleq Hash(LL1NVRAM.historySummary, stateHash)$

The *LL1NVRAMHistorySummaryUncorrupted* predicate has two conditions. First, that there exists a state hash in *HashType*, for which we have a witness.

$\langle 2 \rangle 4.$ $stateHash \in HashType$

BY $\langle 2 \rangle 3$

The second condition is that the history state binding is authenticated in the unprimed state.

$\langle 2 \rangle 5.$ $LL1HistoryStateBindingAuthenticated(historyStateBinding)$

We will use the assumption that there is no change to the authentication status of any history state binding.

$\langle 3 \rangle 1.$ $\forall\, historyStateBinding1 \in HashType$:

UNCHANGED $LL1HistoryStateBindingAuthenticated(historyStateBinding1)$

BY $\langle 1 \rangle 1$

This requires that we prove the type of the history state binding.

$\langle 3 \rangle 2.$ $historyStateBinding \in HashType$

$\langle 4 \rangle 1.$ $LL1NVRAM.historySummary \in HashDomain$

$\langle 5 \rangle 1.$ $LL1NVRAM.historySummary \in HashType$

$\langle 6 \rangle 1.$ $LL1TypeInvariant$

BY $\langle 1 \rangle 1$

$\langle 6 \rangle 2.$ QED

BY $\langle 6 \rangle 1$, *LL1SubtypeImplicationLemma*DEF *LL1SubtypeImplication*

$\langle 5 \rangle 2.$ QED

BY $\langle 5 \rangle 1$ DEF *HashDomain*

$\langle 4 \rangle 2.$ $stateHash \in HashDomain$

BY $\langle 2 \rangle 4$ DEF *HashDomain*

$\langle 4 \rangle 3.$ QED

BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$, *HashTypeSafe*DEF *historyStateBinding*

We know, in our contradictory universe, that the history state binding is authenticated in the primed state.

$\langle 3 \rangle 3.$ $LL1HistoryStateBindingAuthenticated(historyStateBinding)'$

BY $\langle 2 \rangle 3$

By applying the assumption that there is no change to the authentication status of any history state binding, we show that the history state binding is authenticated in the unprimed state.

$\langle 3 \rangle 4.$ QED
BY $\langle 3 \rangle 1$, $\langle 3 \rangle 2$, $\langle 3 \rangle 3$

Since both conditions are satisfied, the *LL1NVRAMHistorySummaryUncorrupted* predicate is true in the unprimed state.

$\langle 2 \rangle 6.$ $LL1NVRAMHistorySummaryUncorrupted$
BY $\langle 2 \rangle 4$, $\langle 2 \rangle 5$ DEF $LL1NVRAMHistorySummaryUncorrupted$

However, we are considering the case in which the *LL1NVRAMHistorySummaryUncorrupted* predicate is false in the unprimed state, so we have a contradiction.

$\langle 2 \rangle 7.$ QED
BY $\langle 1 \rangle 3$, $\langle 2 \rangle 6$

The predicate has a boolean truth value.

$\langle 1 \rangle 4.$ $LL1NVRAMHistorySummaryUncorrupted \in$ BOOLEAN
BY DEF $LL1NVRAMHistorySummaryUncorrupted$

$\langle 1 \rangle 5.$ QED
BY $\langle 1 \rangle 2$, $\langle 1 \rangle 3$, $\langle 1 \rangle 4$

The *LL1RepeatOperationUnchangedObseredOutputsLemma* states that the *LL1RepeatOperation* action does not change the value of *LL1ObservedOutputs*. Together with the *LL1RepeatOperationUnchangedAuthenticatedHistoryStateBindingsLemma* below, this is the essence of why *LL1RepeatOperation* does not cause any change to the refined high-level state.

THEOREM $LL1RepeatOperationUnchangedObseredOutputsLemma \triangleq$
$LL1TypeInvariant \land UnforgeabilityInvariant \land InclusionInvariant \land LL1RepeatOperation \Rightarrow$
UNCHANGED $LL1ObservedOutputs$

First, we assume the antecedents.

$\langle 1 \rangle 1.$ HAVE $LL1TypeInvariant \land UnforgeabilityInvariant \land InclusionInvariant \land LL1RepeatOperation$

Then, we pick some input for which *LL1RepeatOperation* is true.

$\langle 1 \rangle 2.$ PICK $input \in LL1AvailableInputs : LL1RepeatOperation!(input)!1$
BY $\langle 1 \rangle 1$ DEF $LL1RepeatOperation$

To simplify the writing of the proof, we re-state some of the definitions from the *LL1RepeatOperation* action.

$\langle 1 \rangle$ $stateHash \triangleq Hash(LL1RAM.publicState, LL1RAM.privateStateEnc)$
$\langle 1 \rangle$ $historyStateBinding \triangleq Hash(LL1RAM.historySummary, stateHash)$
$\langle 1 \rangle$ $privateState \triangleq SymmetricDecrypt(LL1NVRAM.symmetricKey, LL1RAM.privateStateEnc)$
$\langle 1 \rangle$ $sResult \triangleq Service(LL1RAM.publicState, privateState, input)$

We then assert the type safety of these definitions, with the help of the *LL1RepeatOperationDefsTypeSafeLemma*.

$\langle 1 \rangle 3.$ $\land$ $stateHash \in HashType$
$\land$ $historyStateBinding \in HashType$
$\land$ $privateState \in PrivateStateType$
$\land$ $sResult \in ServiceResultType$
$\land$ $sResult.newPublicState \in PublicStateType$
$\land$ $sResult.newPrivateState \in PrivateStateType$
$\land$ $sResult.output \in OutputType$

$\langle 2 \rangle 1.$ $input \in LL1AvailableInputs$
BY $\langle 1 \rangle 2$

$\langle 2 \rangle 2.$ $LL1TypeInvariant$
BY $\langle 1 \rangle 1$

$\langle 2 \rangle 3.$ QED

BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, *LL1RepeatOperationDefsTypeSafeLemma*

We hide the definitions, so they don't overwhelm the prover. We'll pull them in as necessary below.

$\langle 1 \rangle$ HIDE DEF *stateHash*, *historyStateBinding*, *privateState*, *sResult*

From the definition of *LL1RepeatOperation*, we see the primed state of *LL1ObservedOutputs* is formed by unioning in the output from the service.

$\langle 1 \rangle 4.$ $LL1ObservedOutputs' = LL1ObservedOutputs \cup \{sResult.output\}$
BY $\langle 1 \rangle 2$ DEF *LL1RepeatOperation*, *sResult*, *privateState*

We then show that the output from the service is already in *LL1ObservedOutputs*.

$\langle 1 \rangle 5.$ $sResult.output \in LL1ObservedOutputs$

Our strategy is to use the *InclusionInvariant*. We first have to show that all of the types are satisfied.

$\langle 2 \rangle 1.$ *LL1TypeInvariant*
BY $\langle 1 \rangle 1$
$\langle 2 \rangle 2.$ $input \in InputType$
$\langle 3 \rangle 1.$ $input \in LL1AvailableInputs$
BY $\langle 1 \rangle 2$
$\langle 3 \rangle 2.$ $LL1AvailableInputs \subseteq InputType$
BY $\langle 2 \rangle 1$ DEF *LL1TypeInvariant*
$\langle 3 \rangle 3.$ QED
BY $\langle 3 \rangle 1$, $\langle 3 \rangle 2$
$\langle 2 \rangle 3.$ $\land$ $LL1RAM.historySummary \in HashType$
$\land$ $LL1RAM.publicState \in PublicStateType$
$\land$ $LL1RAM.privateStateEnc \in PrivateStateEncType$
BY $\langle 2 \rangle 1$, *LL1SubtypeImplicationLemma*DEF *LL1SubtypeImplication*

We then have to show that the antecedents in the *InclusionInvariant* are satisfied.

$\langle 2 \rangle 4.$ $LL1NVRAM.historySummary = Hash(LL1RAM.historySummary, input)$
BY $\langle 1 \rangle 2$
$\langle 2 \rangle 5.$ $LL1HistoryStateBindingAuthenticated(historyStateBinding)$

To show that the history state binding is authenticated, we demonstrate that *LL1RAM.authenticator* is a sufficient witness for the existential quantifier within the definition of *LL1HistoryStateBindingAuthenticated*.

$\langle 3 \rangle 1.$ $ValidateMAC(LL1NVRAM.symmetricKey, historyStateBinding, LL1RAM.authenticator)$
BY $\langle 1 \rangle 2$ DEF *historyStateBinding*, *stateHash*
$\langle 3 \rangle 2.$ $LL1RAM.authenticator \in LL1ObservedAuthenticators$
$\langle 4 \rangle 1.$ $historyStateBinding \in HashType$
BY $\langle 1 \rangle 3$
$\langle 4 \rangle 2.$ *UnforgeabilityInvariant*
BY $\langle 1 \rangle 1$
$\langle 4 \rangle 3.$ QED
BY $\langle 3 \rangle 1$, $\langle 4 \rangle 1$, $\langle 4 \rangle 2$ DEF *UnforgeabilityInvariant*
$\langle 3 \rangle 3.$ QED
BY $\langle 3 \rangle 1$, $\langle 3 \rangle 2$ DEF *LL1HistoryStateBindingAuthenticated*

Then, we can apply the *InclusionInvariant* to show that the output from the service is in the set of observed outputs.

$\langle 2 \rangle 6.$ QED
$\langle 3 \rangle 1.$ *InclusionInvariant*
BY $\langle 1 \rangle 1$
$\langle 3 \rangle 2.$ QED
BY $\langle 2 \rangle 2$, $\langle 2 \rangle 3$, $\langle 2 \rangle 4$, $\langle 2 \rangle 5$, $\langle 3 \rangle 1$
DEF *InclusionInvariant*, *sResult*, *privateState*, *historyStateBinding*, *stateHash*

Since the element being uniond into the set is already in the set, the set does not change.

$\langle 1 \rangle 6.$ QED
BY $\langle 1 \rangle 4$, $\langle 1 \rangle 5$

The $LL1RepeatOperationUnchangedAuthenticatedHistoryStateBindingsLemma$ states that the $LL1RepeatOperation$ action does not change the set of history state bindings that have authenticators in the set $LL1ObservedAuthenticators$. Together with the $LL1RepeatOperationUnchangedObseredOutputsLemma$ above, this is the essence of why $LL1RepeatOperation$ does not cause any change to the refined high-level state.

THEOREM $LL1RepeatOperationUnchangedAuthenticatedHistoryStateBindingsLemma \triangleq$
$LL1TypeInvariant \land UnforgeabilityInvariant \land InclusionInvariant \land LL1RepeatOperation \Rightarrow$
$\forall\, historyStateBinding1 \in HashType :$
UNCHANGED $LL1HistoryStateBindingAuthenticated(historyStateBinding1)$

We assume the antecedents.

⟨1⟩1. HAVE $LL1TypeInvariant \land UnforgeabilityInvariant \land InclusionInvariant \land LL1RepeatOperation$

To prove the universally quantified expression, we take a new history state binding in $HashType$.

⟨1⟩2. TAKE $historyStateBinding1 \in HashType$

Then, we pick some input for which $LL1RepeatOperation$ is true.

⟨1⟩3. PICK $input \in LL1AvailableInputs : LL1RepeatOperation!(input)!1$
BY ⟨1⟩1 DEF $LL1RepeatOperation$

To simplify the writing of the proof, we re-state the definitions from the $LL1RepeatOperation$ action.

⟨1⟩ $stateHash \triangleq Hash(LL1RAM.publicState,\ LL1RAM.privateStateEnc)$
⟨1⟩ $historyStateBinding \triangleq Hash(LL1RAM.historySummary,\ stateHash)$
⟨1⟩ $privateState \triangleq SymmetricDecrypt(LL1NVRAM.symmetricKey,\ LL1RAM.privateStateEnc)$
⟨1⟩ $sResult \triangleq Service(LL1RAM.publicState,\ privateState,\ input)$
⟨1⟩ $newPrivateStateEnc \triangleq$
$SymmetricEncrypt(LL1NVRAM.symmetricKey,\ sResult.newPrivateState)$
⟨1⟩ $newStateHash \triangleq Hash(sResult.newPublicState,\ newPrivateStateEnc)$
⟨1⟩ $newHistoryStateBinding \triangleq Hash(LL1NVRAM.historySummary,\ newStateHash)$
⟨1⟩ $newAuthenticator \triangleq GenerateMAC(LL1NVRAM.symmetricKey,\ newHistoryStateBinding)$

We then assert the type safety of these definitions, with the help of the $LL1RepeatOperationDefsTypeSafeLemma$.

⟨1⟩4. $\land\ stateHash \in HashType$
$\land\ historyStateBinding \in HashType$
$\land\ privateState \in PrivateStateType$
$\land\ sResult \in ServiceResultType$
$\land\ sResult.newPublicState \in PublicStateType$
$\land\ sResult.newPrivateState \in PrivateStateType$
$\land\ sResult.output \in OutputType$
$\land\ newPrivateStateEnc \in PrivateStateEncType$
$\land\ newStateHash \in HashType$
$\land\ newHistoryStateBinding \in HashType$
$\land\ newAuthenticator \in MACType$
⟨2⟩1. $input \in LL1AvailableInputs$
BY ⟨1⟩3
⟨2⟩2. $LL1TypeInvariant$
BY ⟨1⟩1
⟨2⟩3. QED
BY ⟨2⟩1, ⟨2⟩2, $LL1RepeatOperationDefsTypeSafeLemma$

We hide the definitions, so they don't overwhelm the prover. We'll pull them in as necessary below.

⟨1⟩ HIDE DEF $stateHash$, $historyStateBinding$, $privateState$, $sResult$, $newPrivateStateEnc$,
$newStateHash$, $newHistoryStateBinding$, $newAuthenticator$

From the definition of $LL1RepeatOperation$, we see the primed state of $LL1ObservedAuthenticators$ is formed by unioning in the new authenticator.

$\langle 1 \rangle 5.$ $LL1ObservedAuthenticators' =$
$LL1ObservedAuthenticators \cup \{newAuthenticator\}$
BY $\langle 1 \rangle 3$ DEF *LL1RepeatOperation*, *newAuthenticator*, *newHistoryStateBinding*, *newStateHash*, *newPrivateStateEnc*, *sResult*, *privateState*

One fact that will be useful in several places is that the symmetric key in the *NVRAM* has not changed.

$\langle 1 \rangle 6.$ UNCHANGED $LL1NVRAM.symmetricKey$
$\langle 2 \rangle 1.$ UNCHANGED $LL1NVRAM$
BY $\langle 1 \rangle 3$
$\langle 2 \rangle 2.$ QED
BY $\langle 2 \rangle 1$

To show that *LL1HistoryStateBindingAuthenticated* predicate is unchanged for all history state bindings, we first consider one specific history state binding, namely the the new history state binding defined in *LL1RepeatOperation*.

$\langle 1 \rangle 7.$ CASE $historyStateBinding1 = newHistoryStateBinding$

First, we'll show that the new history state binding is authenticated in the unprimed state.

$\langle 2 \rangle 1.$ $LL1HistoryStateBindingAuthenticated(newHistoryStateBinding)$

Our strategy is to use the *InclusionInvariant*. We first have to show that all of the types are satisfied.

$\langle 3 \rangle 2.$ $LL1TypeInvariant$
BY $\langle 1 \rangle 1$
$\langle 3 \rangle 3.$ $input \in InputType$
$\langle 4 \rangle 1.$ $input \in LL1AvailableInputs$
BY $\langle 1 \rangle 3$
$\langle 4 \rangle 2.$ $LL1AvailableInputs \subseteq InputType$
BY $\langle 3 \rangle 2$ DEF *LL1TypeInvariant*
$\langle 4 \rangle 3.$ QED
BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$
$\langle 3 \rangle 4.$ $\wedge\ LL1RAM.historySummary \in HashType$
$\wedge\ LL1RAM.publicState \in PublicStateType$
$\wedge\ LL1RAM.privateStateEnc \in PrivateStateEncType$
BY $\langle 3 \rangle 2$, *LL1SubtypeImplicationLemma*DEF *LL1SubtypeImplication*

We then have to show that the antecedents in the *InclusionInvariant* are satisfied.

$\langle 3 \rangle 5.$ $LL1NVRAM.historySummary = Hash(LL1RAM.historySummary, input)$
BY $\langle 1 \rangle 3$
$\langle 3 \rangle 6.$ $LL1HistoryStateBindingAuthenticated(historyStateBinding)$

To show that the history state binding is authenticated, we demonstrate that *LL1RAM.authenticator* is a sufficient witness for the existential quantifier within the definition of *LL1HistoryStateBindingAuthenticated*.

$\langle 4 \rangle 1.$ $ValidateMAC(LL1NVRAM.symmetricKey, historyStateBinding, LL1RAM.authenticator)$
BY $\langle 1 \rangle 3$ DEF *historyStateBinding*, *stateHash*
$\langle 4 \rangle 2.$ $LL1RAM.authenticator \in LL1ObservedAuthenticators$
$\langle 5 \rangle 1.$ $historyStateBinding \in HashType$
BY $\langle 1 \rangle 4$
$\langle 5 \rangle 2.$ $UnforgeabilityInvariant$
BY $\langle 1 \rangle 1$
$\langle 5 \rangle 3.$ QED
BY $\langle 4 \rangle 1$, $\langle 5 \rangle 1$, $\langle 5 \rangle 2$ DEF *UnforgeabilityInvariant*
$\langle 4 \rangle 3.$ QED
BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$ DEF *LL1HistoryStateBindingAuthenticated*

Then, we can apply the *InclusionInvariant* to show that the new history state binding is authenticated.

$\langle 3 \rangle 7.$ QED
$\langle 4 \rangle 1.$ $InclusionInvariant$
BY $\langle 1 \rangle 1$

$\langle 4 \rangle 2.$ QED
BY $\langle 3 \rangle 3$, $\langle 3 \rangle 4$, $\langle 3 \rangle 5$, $\langle 3 \rangle 6$, $\langle 4 \rangle 1$
DEF *InclusionInvariant*, *newAuthenticator*, *newHistoryStateBinding*, *newStateHash*, *newPrivateStateEnc*, *sResult*, *privateState*, *historyStateBinding*, *stateHash*

Next, we'll show that the new history state binding is authenticated in the primed state.

$\langle 2 \rangle 2.$ $LL1HistoryStateBindingAuthenticated(historyStateBinding1)'$

By expanding the definition of *LL1HistoryStateBindingAuthenticated*, it suffices to show that the new authenticator defined in *LL1RepeatOperation* (which we know to be in the primed set of observed authenticators) is a valid *MAC* for the history state binding in the primed state.

$\langle 3 \rangle 1.$ SUFFICES $ValidateMAC(LL1NVRAM.symmetricKey', historyStateBinding1, newAuthenticator)$
$\langle 4 \rangle 1.$ $newAuthenticator \in LL1ObservedAuthenticators'$
BY $\langle 1 \rangle 5$
$\langle 4 \rangle 2.$ QED
BY $\langle 4 \rangle 1$ DEF *LL1HistoryStateBindingAuthenticated*

The new authenticator was genererated as a *MAC* of this history state binding by *LL1RepeatOperation* in the unprimed state, and the symmetric key in the *NVRAM* has not changed.

$\langle 3 \rangle 2.$ $newAuthenticator = GenerateMAC(LL1NVRAM.symmetricKey', historyStateBinding1)$
BY $\langle 1 \rangle 6$, $\langle 1 \rangle 7$ DEF *newAuthenticator*

We can thus use the *MACComplete* property to show that the generated *MAC* validates appropriately. To do this, we first need to prove some types.

$\langle 3 \rangle 3.$ $LL1NVRAM.symmetricKey' \in SymmetricKeyType$
$\langle 4 \rangle 1.$ $LL1NVRAM.symmetricKey \in SymmetricKeyType$
$\langle 5 \rangle 1.$ *LL1TypeInvariant*
BY $\langle 1 \rangle 1$
$\langle 5 \rangle 2.$ QED
BY $\langle 5 \rangle 1$, *LL1SubtypeImplicationLemma*DEF *LL1SubtypeImplication*
$\langle 4 \rangle 2.$ QED
BY $\langle 1 \rangle 6$, $\langle 4 \rangle 1$
$\langle 3 \rangle 4.$ $historyStateBinding1 \in HashType$
BY $\langle 1 \rangle 2$

Then, we appeal to the *MACComplete* property in a straightforward way.

$\langle 3 \rangle 5.$ QED
BY $\langle 3 \rangle 2$, $\langle 3 \rangle 3$, $\langle 3 \rangle 4$, *MACComplete*

Because the new history state binding is authenticated in both the unprimed and primed states, the *LL1HistoryStateBindingAuthenticated* is unchanged for this history state binding.

$\langle 2 \rangle 3.$ QED
BY $\langle 1 \rangle 7$, $\langle 2 \rangle 1$, $\langle 2 \rangle 2$

We then consider every state binding that is not equal to the the new history state binding defined in *LL1RepeatOperation*.

$\langle 1 \rangle 8.$ CASE $historyStateBinding1 \neq newHistoryStateBinding$

We'll subdivide these into two cases. In the first case, we'll consider the history state bindings that are authenticated in the unprimed state, and we'll show that they continue to be authenticated in the primed state.

$\langle 2 \rangle 1.$ CASE $LL1HistoryStateBindingAuthenticated(historyStateBinding1) = \text{TRUE}$
$\langle 3 \rangle 1.$ $LL1HistoryStateBindingAuthenticated(historyStateBinding1)' = \text{TRUE}$

By hypothesis, the history state binding is authenticated in the unprimed state. Thus, we can pick an authenticator in the set of observed authenticators that is a valid *MAC* for this history state binding.

$\langle 4 \rangle 1.$ PICK $authenticator \in LL1ObservedAuthenticators :$
$ValidateMAC(LL1NVRAM.symmetricKey, historyStateBinding1, authenticator)$
BY $\langle 2 \rangle 1$ DEF *LL1HistoryStateBindingAuthenticated*

Because the symmetric key in the $NVRAM$ has not changed, this authenticator is also a valid MAC for this history state binding in the primed state.

$\langle 4 \rangle 2$. $ValidateMAC(LL1NVRAM.symmetricKey', historyStateBinding1, authenticator)$
BY $\langle 1 \rangle 6$, $\langle 4 \rangle 1$

Because the primed set of observed authenticators includes all authenticators that were in the unprimed set, this authenticator is also in the primed set of observed authenticators.

$\langle 4 \rangle 3$. $authenticator \in LL1ObservedAuthenticators'$
$\langle 5 \rangle 1$. $authenticator \in LL1ObservedAuthenticators$
BY $\langle 4 \rangle 1$
$\langle 5 \rangle 2$. $LL1ObservedAuthenticators \subseteq LL1ObservedAuthenticators'$
BY $\langle 1 \rangle 5$
$\langle 5 \rangle 3$. QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$

The previous two conditions are sufficient to establish that the history state binding is authenticated in the primed state.

$\langle 4 \rangle 4$. QED
BY $\langle 4 \rangle 2$, $\langle 4 \rangle 3$ DEF $LL1HistoryStateBindingAuthenticated$

Because the history state binding is authenticated in both the unprimed and primed states, the $LL1HistoryStateBindingAuthenticated$ is unchanged for this history state binding.

$\langle 3 \rangle 2$. QED
BY $\langle 2 \rangle 1$, $\langle 3 \rangle 1$

We'll subdivide these into two cases. In the second case, we'll consider the history state bindings that are unauthenticated in the unprimed state, and we'll show that they continue to be unauthenticated in the primed state.

$\langle 2 \rangle 2$. CASE $LL1HistoryStateBindingAuthenticated(historyStateBinding1) = \text{FALSE}$
$\langle 3 \rangle 1$. $LL1HistoryStateBindingAuthenticated(historyStateBinding1)' = \text{FALSE}$

To prove that the history state binding is not authenticated in the primed state, it suffices to show that none of the authenticators in the primed set of observed authenticators is a valid MAC for the history state binding.

$\langle 4 \rangle 1$. SUFFICES $\forall\, authenticator \in LL1ObservedAuthenticators'$:
$\neg ValidateMAC(LL1NVRAM.symmetricKey', historyStateBinding1, authenticator)$
BY DEF $LL1HistoryStateBindingAuthenticated$

To prove the universally quantified expression, we take a new authenticator in the primed set of observed authenticators.

$\langle 4 \rangle 2$. TAKE $authenticator \in LL1ObservedAuthenticators'$

We'll subdivide this into two cases. First, we consider the case in which the authenticator is in the unprimed set of authenticators. In this case, because the authenticator failed to authenticate the history state binding in the unprimed state, and the symmetric key has not changed, it immediately follows that the authenticator will not authenticate the history state binding in the primed state.

$\langle 4 \rangle 3$. CASE $authenticator \in LL1ObservedAuthenticators$
BY $\langle 1 \rangle 6$, $\langle 2 \rangle 2$, $\langle 4 \rangle 3$ DEF $LL1HistoryStateBindingAuthenticated$

In the second case, we consider the new authenticator defined in $LL1RepeatOperation$.

$\langle 4 \rangle 4$. CASE $authenticator = newAuthenticator$

We'll use proof by contradiction. Assume that the new authenticator is a valid MAC for the history state binding.

$\langle 5 \rangle 1$. SUFFICES
ASSUME $ValidateMAC(LL1NVRAM.symmetricKey', historyStateBinding1, authenticator)$
PROVE FALSE
OBVIOUS

By the collision resistance of $MACs$, it must be the case the history state binding is equal to the new history state binding defined in $LL1RepeatOperation$.

$\langle 5 \rangle 2$. $historyStateBinding1 = newHistoryStateBinding$

$\langle 6 \rangle 1.$ $LL1NVRAM.symmetricKey \in SymmetricKeyType$
$\langle 7 \rangle 1.$ $LL1TypeInvariant$
BY $\langle 1 \rangle 1$
$\langle 7 \rangle 2.$ QED
BY $\langle 7 \rangle 1$, $LL1SubtypeImplicationLemma$ DEF $LL1SubtypeImplication$
$\langle 6 \rangle 2.$ $LL1NVRAM.symmetricKey' \in SymmetricKeyType$
BY $\langle 1 \rangle 6$, $\langle 6 \rangle 1$
$\langle 6 \rangle 3.$ $historyStateBinding1 \in HashType$
BY $\langle 1 \rangle 2$
$\langle 6 \rangle 4.$ $newHistoryStateBinding \in HashType$
BY $\langle 1 \rangle 4$
$\langle 6 \rangle 5.$ QED
BY $\langle 4 \rangle 4$, $\langle 5 \rangle 1$, $\langle 6 \rangle 1$, $\langle 6 \rangle 2$, $\langle 6 \rangle 3$, $\langle 6 \rangle 4$, $MACCollisionResistant$ DEF $newAuthenticator$

But we are working within a case in which the history state binding is not equal to the new history state binding defined in *LL1RepeatOperation*. Thus, we have a contradiction.

$\langle 5 \rangle 3.$ QED
BY $\langle 1 \rangle 8$, $\langle 5 \rangle 2$

We've considered authenticators in the unprimed set of authenticators, and we've considered the new authenticator defined in *LL1RepeatOperation*. Because the primed set of authenticators is the union of these two, we have exhausted the cases.

$\langle 4 \rangle 5.$ QED
BY $\langle 1 \rangle 5$, $\langle 4 \rangle 3$, $\langle 4 \rangle 4$

Because the history state binding is unauthenticated in both the unprimed and primed states, the *LL1HistoryStateBindingAuthenticated* is unchanged for this history state binding.

$\langle 3 \rangle 2.$ QED
BY $\langle 2 \rangle 2$, $\langle 3 \rangle 1$

By proving that *LL1HistoryStateBindingAuthenticated* is a boolean predicate, it is immediately clear that the two cases of true and false are exhaustive for this predicate.

$\langle 2 \rangle 3.$ $LL1HistoryStateBindingAuthenticated(historyStateBinding1) \in$ BOOLEAN
BY DEF $LL1HistoryStateBindingAuthenticated$
$\langle 2 \rangle 4.$ QED
BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, $\langle 2 \rangle 3$

Because the conclusion holds for (1) the new history state binding defined in *LL1RepeatOperation* and (2) every other state binding, the conclusion holds for all state bindings.

$\langle 1 \rangle 9.$ QED
BY $\langle 1 \rangle 7$, $\langle 1 \rangle 8$

The *LL1RAMUnforgeabilityUnchangedLemma* states that, if the symmetric key in the *NVRAM* does not change and the set of observed authenticators does not change, then the RAM's portion of the *ExtendedUnforgeabilityInvariant* inductively holds when the RAM's primed value is taken from the RAM or the disk.

THEOREM $LL1RAMUnforgeabilityUnchangedLemma \triangleq$

$$
\begin{array}{l}
(\land\ ExtendedUnforgeabilityInvariant \\
\ \ \land\ LL1TypeInvariant \\
\ \ \land\ LL1TypeInvariant' \\
\ \ \land\ LL1RAM' \in \{LL1RAM,\ LL1Disk\} \\
\ \ \land\ LL1ObservedAuthenticators \subseteq LL1ObservedAuthenticators' \\
\ \ \land\ \text{UNCHANGED}\ LL1NVRAM.symmetricKey) \\
\Rightarrow \\
\quad \forall\, historyStateBinding \in HashType : \\
\quad\quad ValidateMAC(LL1NVRAM.symmetricKey',\ historyStateBinding,\ LL1RAM.authenticator') \Rightarrow
\end{array}
$$

$LL1RAM.authenticator' \in LL1ObservedAuthenticators'$

We begin by assuming the antecedent.

$\langle 1 \rangle 1.$ HAVE $\land$ *ExtendedUnforgeabilityInvariant*
$\land$ *LL1TypeInvariant*
$\land$ *LL1TypeInvariant'*
$\land$ $LL1RAM' \in \{LL1RAM, LL1Disk\}$
$\land$ $LL1ObservedAuthenticators \subseteq LL1ObservedAuthenticators'$
$\land$ UNCHANGED *LL1NVRAM.symmetricKey*

To prove the universally quantified expression, we take a new *historyStateBinding* in the *HashType*.

$\langle 1 \rangle 2.$ TAKE $historyStateBinding \in HashType$

We then assume the antecedent in the nested implication.

$\langle 1 \rangle 3.$ HAVE $ValidateMAC(LL1NVRAM.symmetricKey', historyStateBinding, LL1RAM.authenticator')$

The *LL1RAM*'s primed state is taken from either the *LL1RAM* or the disk.

$\langle 1 \rangle 4.$ $LL1RAM' \in \{LL1RAM, LL1Disk\}$
BY $\langle 1 \rangle 1$

Case 1: the *LL1RAM*'s primed state comes from the *LL1RAM*. There are three basic steps.

$\langle 1 \rangle 5.$ CASE UNCHANGED *LL1RAM*

First, since we take the antecedent in the nested implication and swap out unprimed variables for primed variables, since the symmetric key and authenticator have not changed.

$\langle 2 \rangle 1.$ $ValidateMAC(LL1NVRAM.symmetricKey, historyStateBinding, LL1RAM.authenticator)$
$\langle 3 \rangle 1.$ UNCHANGED *LL1NVRAM.symmetricKey*
BY $\langle 1 \rangle 1$
$\langle 3 \rangle 2.$ UNCHANGED *LL1RAM.authenticator*
BY $\langle 1 \rangle 5$
$\langle 3 \rangle 3.$ QED
BY $\langle 1 \rangle 3$, $\langle 3 \rangle 1$, $\langle 3 \rangle 2$

Second, using the *ExtendedUnforgeabilityInvariant*, we show that the authenticator was in the unprimed set of observed authenticators.

$\langle 2 \rangle 2.$ $LL1RAM.authenticator \in LL1ObservedAuthenticators$
$\langle 3 \rangle 1.$ *ExtendedUnforgeabilityInvariant*
BY $\langle 1 \rangle 1$
$\langle 3 \rangle 2.$ QED
BY $\langle 1 \rangle 2$, $\langle 2 \rangle 1$, $\langle 3 \rangle 1$ DEF *ExtendedUnforgeabilityInvariant*

Third, we show that the authenticator is also in the primed set of observed authenticators, since the symmetric key has not changed and set of observed authenticators includes every element in the primed state that it included in the unprimed state.

$\langle 2 \rangle 3.$ QED
$\langle 3 \rangle 1.$ UNCHANGED *LL1RAM.authenticator*
BY $\langle 1 \rangle 5$
$\langle 3 \rangle 2.$ $LL1ObservedAuthenticators \subseteq LL1ObservedAuthenticators'$
BY $\langle 1 \rangle 1$
$\langle 3 \rangle 3.$ QED
BY $\langle 2 \rangle 2$, $\langle 3 \rangle 1$, $\langle 3 \rangle 2$

Case 2: the RAM's primed state comes from the disk. The proof is straightforward.

$\langle 1 \rangle 6.$ CASE $LL1RAM' = LL1Disk$

First, since we take the antecedent in the nested implication and make two changes: (1) swap out unprimed variables for primed variables and (2) replace *LL1RAM* with *LL1Disk*, since the primed state of the RAM comes from the unprimed state of the disk.

$\langle 2 \rangle 1.$ $ValidateMAC(LL1NVRAM.symmetricKey, historyStateBinding, LL1Disk.authenticator)$
$\langle 3 \rangle 1.$ UNCHANGED *LL1NVRAM.symmetricKey*

BY $\langle 1 \rangle 1$
$\langle 3 \rangle 2.$ $LL1RAM.authenticator' = LL1Disk.authenticator$
BY $\langle 1 \rangle 6$
$\langle 3 \rangle 3.$ QED
BY $\langle 1 \rangle 3$, $\langle 3 \rangle 1$, $\langle 3 \rangle 2$

Second, using the *ExtendedUnforgeabilityInvariant*, we show that the authenticator was in the unprimed set of observed authenticators.

$\langle 2 \rangle 2.$ $LL1Disk.authenticator \in LL1ObservedAuthenticators$
$\langle 3 \rangle 1.$ $ExtendedUnforgeabilityInvariant$
BY $\langle 1 \rangle 1$
$\langle 3 \rangle 2.$ QED
BY $\langle 1 \rangle 2$, $\langle 2 \rangle 1$, $\langle 3 \rangle 1$ DEF $ExtendedUnforgeabilityInvariant$

Third, we show that the authenticator is also in the primed set of observed authenticators, since the symmetric key has not changed and set of observed authenticators includes every element in the primed state that it included in the unprimed state.

$\langle 2 \rangle 3.$ QED
$\langle 3 \rangle 1.$ $LL1RAM.authenticator' = LL1Disk.authenticator$
BY $\langle 1 \rangle 6$
$\langle 3 \rangle 2.$ $LL1ObservedAuthenticators \subseteq LL1ObservedAuthenticators'$
BY $\langle 1 \rangle 1$
$\langle 3 \rangle 3.$ QED
BY $\langle 2 \rangle 2$, $\langle 3 \rangle 1$, $\langle 3 \rangle 2$

The theorem is true by exhaustive case analysis.

$\langle 1 \rangle 7.$ QED
BY $\langle 1 \rangle 4$, $\langle 1 \rangle 5$, $\langle 1 \rangle 6$

The *LL1DiskUnforgeabilityUnchangedLemma* states that, if the symmetric key in the *NVRAM* does not change and the set of observed authenticators does not change, then the disk's portion of the *ExtendedUnforgeabilityInvariant* inductively holds when the disk's primed value is taken from the RAM or the disk.

THEOREM $LL1DiskUnforgeabilityUnchangedLemma \triangleq$

$$
\begin{array}{l}
(\land\ ExtendedUnforgeabilityInvariant \\
\ \land\ LL1TypeInvariant \\
\ \land\ LL1TypeInvariant' \\
\ \land\ LL1Disk' \in \{LL1RAM,\ LL1Disk\} \\
\ \land\ LL1ObservedAuthenticators \subseteq LL1ObservedAuthenticators' \\
\ \land\ \text{UNCHANGED } LL1NVRAM.symmetricKey) \\
\Rightarrow \\
\quad \forall\, historyStateBinding \in HashType : \\
\qquad ValidateMAC(LL1NVRAM.symmetricKey',\ historyStateBinding,\ LL1Disk.authenticator') \Rightarrow \\
\qquad\quad LL1Disk.authenticator' \in LL1ObservedAuthenticators'
\end{array}
$$

We begin by assuming the antecedent.

$\langle 1 \rangle 1.$ HAVE $\land\ ExtendedUnforgeabilityInvariant$
$\land\ LL1TypeInvariant$
$\land\ LL1TypeInvariant'$
$\land\ LL1Disk' \in \{LL1RAM,\ LL1Disk\}$
$\land\ LL1ObservedAuthenticators \subseteq LL1ObservedAuthenticators'$
$\land$ UNCHANGED $LL1NVRAM.symmetricKey$

To prove the universally quantified expression, we take a new *historyStateBinding* in the *HashType*.

$\langle 1 \rangle 2.$ TAKE $historyStateBinding \in HashType$

We then assume the antecedent in the nested implication.

$\langle 1 \rangle 3.$ HAVE $\mathit{ValidateMAC}(\mathit{LL1NVRAM}.\mathit{symmetricKey}', \mathit{historyStateBinding}, \mathit{LL1Disk}.\mathit{authenticator}')$

The disk's primed state is taken from either the RAM or the disk.

$\langle 1 \rangle 4.$ $\mathit{LL1Disk}' \in \{\mathit{LL1RAM}, \mathit{LL1Disk}\}$
  BY $\langle 1 \rangle 1$

Case 1: the disk's primed state comes from the disk. There are three basic steps.

$\langle 1 \rangle 5.$ CASE UNCHANGED $\mathit{LL1Disk}$

First, since we take the antecedent in the nested implication and swap out unprimed variables for primed variables, since the symmetric key and authenticator have not changed.

$\langle 2 \rangle 1.$ $\mathit{ValidateMAC}(\mathit{LL1NVRAM}.\mathit{symmetricKey}, \mathit{historyStateBinding}, \mathit{LL1Disk}.\mathit{authenticator})$
  $\langle 3 \rangle 1.$ UNCHANGED $\mathit{LL1NVRAM}.\mathit{symmetricKey}$
    BY $\langle 1 \rangle 1$
  $\langle 3 \rangle 2.$ UNCHANGED $\mathit{LL1Disk}.\mathit{authenticator}$
    BY $\langle 1 \rangle 5$
  $\langle 3 \rangle 3.$ QED
    BY $\langle 1 \rangle 3$, $\langle 3 \rangle 1$, $\langle 3 \rangle 2$

Second, using the *ExtendedUnforgeabilityInvariant*, we show that the authenticator was in the unprimed set of observed authenticators.

$\langle 2 \rangle 2.$ $\mathit{LL1Disk}.\mathit{authenticator} \in \mathit{LL1ObservedAuthenticators}$
  $\langle 3 \rangle 1.$ $\mathit{ExtendedUnforgeabilityInvariant}$
    BY $\langle 1 \rangle 1$
  $\langle 3 \rangle 2.$ QED
    BY $\langle 1 \rangle 2$, $\langle 2 \rangle 1$, $\langle 3 \rangle 1$ DEF $\mathit{ExtendedUnforgeabilityInvariant}$

Third, we show that the authenticator is also in the primed set of observed authenticators, since the symmetric key has not changed and set of observed authenticators includes every element in the primed state that it included in the unprimed state.

$\langle 2 \rangle 3.$ QED
  $\langle 3 \rangle 1.$ UNCHANGED $\mathit{LL1Disk}.\mathit{authenticator}$
    BY $\langle 1 \rangle 5$
  $\langle 3 \rangle 2.$ $\mathit{LL1ObservedAuthenticators} \subseteq \mathit{LL1ObservedAuthenticators}'$
    BY $\langle 1 \rangle 1$
  $\langle 3 \rangle 3.$ QED
    BY $\langle 2 \rangle 2$, $\langle 3 \rangle 1$, $\langle 3 \rangle 2$

Case 2: the disk's primed state comes from the RAM. The proof is straightforward.

$\langle 1 \rangle 6.$ CASE $\mathit{LL1Disk}' = \mathit{LL1RAM}$

First, since we take the antecedent in the nested implication and make two changes: (1) swap out unprimed variables for primed variables and (2) replace *LL1Disk* with *LL1RAM*, since the primed state of the disk comes from the unprimed state of the RAM.

$\langle 2 \rangle 1.$ $\mathit{ValidateMAC}(\mathit{LL1NVRAM}.\mathit{symmetricKey}, \mathit{historyStateBinding}, \mathit{LL1RAM}.\mathit{authenticator})$
  $\langle 3 \rangle 1.$ UNCHANGED $\mathit{LL1NVRAM}.\mathit{symmetricKey}$
    BY $\langle 1 \rangle 1$
  $\langle 3 \rangle 2.$ $\mathit{LL1Disk}.\mathit{authenticator}' = \mathit{LL1RAM}.\mathit{authenticator}$
    BY $\langle 1 \rangle 6$
  $\langle 3 \rangle 3.$ QED
    BY $\langle 1 \rangle 3$, $\langle 3 \rangle 1$, $\langle 3 \rangle 2$

Second, using the *ExtendedUnforgeabilityInvariant*, we show that the authenticator was in the unprimed set of observed authenticators.

$\langle 2 \rangle 2.$ $\mathit{LL1RAM}.\mathit{authenticator} \in \mathit{LL1ObservedAuthenticators}$
  $\langle 3 \rangle 1.$ $\mathit{ExtendedUnforgeabilityInvariant}$
    BY $\langle 1 \rangle 1$
  $\langle 3 \rangle 2.$ QED
    BY $\langle 1 \rangle 2$, $\langle 2 \rangle 1$, $\langle 3 \rangle 1$ DEF $\mathit{ExtendedUnforgeabilityInvariant}$

Third, we show that the authenticator is also in the primed set of observed authenticators, since the symmetric key has not changed and set of observed authenticators includes every element in the primed state that it included in the unprimed state.

$\langle 2 \rangle 3.$ QED

$\langle 3 \rangle 1.$ $LL1Disk.authenticator' = LL1RAM.authenticator$

BY $\langle 1 \rangle 6$

$\langle 3 \rangle 2.$ $LL1ObservedAuthenticators \subseteq LL1ObservedAuthenticators'$

BY $\langle 1 \rangle 1$

$\langle 3 \rangle 3.$ QED

BY $\langle 2 \rangle 2$, $\langle 3 \rangle 1$, $\langle 3 \rangle 2$

The theorem is true by exhaustive case analysis.

$\langle 1 \rangle 7.$ QED

BY $\langle 1 \rangle 4$, $\langle 1 \rangle 5$, $\langle 1 \rangle 6$

The *InclusionUnchangedLemma* states that, if there are no changes to the *NVRAM*, to the set of observed outputs, or to the authentication status of any history state binding, then the *InclusionInvariant* holds inductively from the unprimed state to the primed state.

THEOREM $InclusionUnchangedLemma \triangleq$

$$
\begin{array}{l}
(\land\ InclusionInvariant \\
\ \land\ LL1TypeInvariant \\
\ \land\ LL1TypeInvariant' \\
\ \land\ \text{UNCHANGED } \langle LL1NVRAM,\ LL1ObservedOutputs \rangle \\
\ \land\ \forall\, historyStateBinding1 \in HashType : \\
\qquad \text{UNCHANGED } LL1HistoryStateBindingAuthenticated(historyStateBinding1)) \\
\Rightarrow \\
\quad InclusionInvariant'
\end{array}
$$

We begin by assuming the antecedent.

$\langle 1 \rangle 1.$ HAVE
$$
\begin{array}{l}
\land\ InclusionInvariant \\
\land\ LL1TypeInvariant \\
\land\ LL1TypeInvariant' \\
\land\ \text{UNCHANGED } \langle LL1NVRAM,\ LL1ObservedOutputs \rangle \\
\land\ \forall\, historyStateBinding1 \in HashType : \\
\qquad \text{UNCHANGED } LL1HistoryStateBindingAuthenticated(historyStateBinding1)
\end{array}
$$

To prove the universally quantified expression, we take a new set of variables in the appropriate types. For the TAKE step to be meaningful to the prover, first we have to tell the prover to expand the definition of *InclusionInvariant*, so it will see the universally quantified expression therein.

$\langle 1 \rangle$ USE DEF $InclusionInvariant$

$\langle 1 \rangle 2.$ TAKE
$$
\begin{array}{l}
input \in InputType, \\
historySummary \in HashType, \\
publicState \in PublicStateType, \\
privateStateEnc \in PrivateStateEncType
\end{array}
$$

To simplify the writing of the proof, we re-state the definitions from the *InclusionInvariant*.

$\langle 1 \rangle$ $stateHash \triangleq Hash(publicState,\ privateStateEnc)$

$\langle 1 \rangle$ $historyStateBinding \triangleq Hash(historySummary,\ stateHash)$

$\langle 1 \rangle$ $privateState \triangleq SymmetricDecrypt(LL1NVRAM.symmetricKey,\ privateStateEnc)$

$\langle 1 \rangle$ $sResult \triangleq Service(publicState,\ privateState,\ input)$

$\langle 1 \rangle$ $newPrivateStateEnc \triangleq$
$\qquad SymmetricEncrypt(LL1NVRAM.symmetricKey,\ sResult.newPrivateState)$

$\langle 1 \rangle$ $newStateHash \triangleq Hash(sResult.newPublicState,\ newPrivateStateEnc)$

$\langle 1 \rangle$ *newHistoryStateBinding* $\triangleq$ $Hash(LL1NVRAM.historySummary, newStateHash)$

We then assert the type safety of these definitions, with the help of the *InclusionInvariantDefsTypeSafeLemma*.

$\langle 1 \rangle 3.$ $\land$ $stateHash \in HashType$
$\land$ $historyStateBinding \in HashType$
$\land$ $privateState \in PrivateStateType$
$\land$ $sResult \in ServiceResultType$
$\land$ $sResult.newPublicState \in PublicStateType$
$\land$ $sResult.newPrivateState \in PrivateStateType$
$\land$ $sResult.output \in OutputType$
$\land$ $newPrivateStateEnc \in PrivateStateEncType$
$\land$ $newStateHash \in HashType$
$\land$ $newHistoryStateBinding \in HashType$

$\langle 2 \rangle 1.$ $LL1TypeInvariant$
BY $\langle 1 \rangle 1$

$\langle 2 \rangle 2.$ QED
BY $\langle 1 \rangle 2$, $\langle 2 \rangle 1$, *InclusionInvariantDefsTypeSafeLemma*

The *InclusionInvariant* states an implication. To prove this, it suffices to assume the antecedent and prove the consequent.

$\langle 1 \rangle 4.$ SUFFICES
ASSUME
$\land$ $LL1NVRAM.historySummary' = Hash(historySummary, input)$
$\land$ $LL1HistoryStateBindingAuthenticated(historyStateBinding)'$
PROVE
$\land$ $sResult.output' \in LL1ObservedOutputs'$
$\land$ $LL1HistoryStateBindingAuthenticated(newHistoryStateBinding)'$
OBVIOUS

We hide the definitions, so they don't overwhelm the prover. We'll pull them in as necessary below.

$\langle 1 \rangle$ HIDE DEF *InclusionInvariant*
$\langle 1 \rangle$ HIDE DEF *stateHash*, *historyStateBinding*, *privateState*, *sResult*, *newPrivateStateEnc*, *newStateHash*, *newHistoryStateBinding*

We will prove each of the conjuncts separately. Following is the proof of the first conjunct.

To prove that the primed output of the service is in the primed set of observed outputs, we prove three things: (1) Before the action, the output of the service is in the set of observed outputs. (2) The output of the service does not change. (3) The set of observed outputs does not change.

$\langle 1 \rangle 5.$ $sResult.output' \in LL1ObservedOutputs'$

Step 1: Before the action, the output of the service is in the set of observed outputs. This follows because the *InclusionInvariant* is true in the unprimed state.

$\langle 2 \rangle 1.$ $sResult.output \in LL1ObservedOutputs$

We prove the two conjuncts in the antecedent of the *InclusionInvariant*. Each follows as a straightforward consequence of the fact that *LL1NVRAM* and *LL1ObservedAuthenticators* have not changed.

$\langle 3 \rangle 1.$ $LL1NVRAM.historySummary = Hash(historySummary, input)$
$\langle 4 \rangle 1.$ $LL1NVRAM.historySummary' = Hash(historySummary, input)$
BY $\langle 1 \rangle 4$
$\langle 4 \rangle 2.$ UNCHANGED $LL1NVRAM.historySummary$
$\langle 5 \rangle 1.$ UNCHANGED $LL1NVRAM$
BY $\langle 1 \rangle 1$
$\langle 5 \rangle 2.$ QED
BY $\langle 5 \rangle 1$
$\langle 4 \rangle 3.$ QED
BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$
$\langle 3 \rangle 2.$ $LL1HistoryStateBindingAuthenticated(historyStateBinding)$

The history state binding is authenticated in the primed state, by hypothesis.

$\langle 4 \rangle 1.$ $LL1HistoryStateBindingAuthenticated(historyStateBinding)'$
BY $\langle 1 \rangle 4$

The authentication status of the history state binding has not changed, because this status has not changed for any history state binding in $HashType$.

$\langle 4 \rangle 2.$ UNCHANGED $LL1HistoryStateBindingAuthenticated(historyStateBinding)$

We have to show that the history state binding has the appropriate type.

$\langle 5 \rangle 1.$ $historyStateBinding \in HashType$
BY $\langle 1 \rangle 3$

The authentication status of all input state bindings has not changed, as assumed by the lemma.

$\langle 5 \rangle 2.$ $\forall\, historyStateBinding1 \in HashType :$
UNCHANGED $LL1HistoryStateBindingAuthenticated(historyStateBinding1)$
BY $\langle 1 \rangle 1$

The conclusion follows directly.

$\langle 5 \rangle 3.$ QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$

Since the history state binding is authenticated in the primed state, and since its authentication status has not changed, it is also authenticated in the unprimed state.

$\langle 4 \rangle 3.$ QED
BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$

We can then use the *InclusionInvariant* to prove that the output of the service is in the set of observed outputs in the unprimed state.

$\langle 3 \rangle 3.$ $InclusionInvariant$
BY $\langle 1 \rangle 1$

$\langle 3 \rangle 4.$ QED
BY $\langle 1 \rangle 2$, $\langle 3 \rangle 1$, $\langle 3 \rangle 2$, $\langle 3 \rangle 3$
DEF $InclusionInvariant$, $sResult$, $privateState$, $historyStateBinding$, $stateHash$

Step 2: The output of the service does not change.

$\langle 2 \rangle 2.$ UNCHANGED $sResult.output$
$\langle 3 \rangle 1.$ UNCHANGED $privateState$
$\langle 4 \rangle 1.$ UNCHANGED $LL1NVRAM.symmetricKey$
$\langle 5 \rangle 1.$ UNCHANGED $LL1NVRAM$
BY $\langle 1 \rangle 1$
$\langle 5 \rangle 2.$ QED
BY $\langle 5 \rangle 1$
$\langle 4 \rangle 2.$ QED
BY $\langle 4 \rangle 1$ DEF $privateState$
$\langle 3 \rangle 2.$ UNCHANGED $sResult$
BY $\langle 3 \rangle 1$ DEF $sResult$
$\langle 3 \rangle 3.$ QED
BY $\langle 3 \rangle 2$

Step 3: The set of observed outputs does not change.

$\langle 2 \rangle 3.$ UNCHANGED $LL1ObservedOutputs$
BY $\langle 1 \rangle 1$
$\langle 2 \rangle 4.$ QED
BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, $\langle 2 \rangle 3$

Following is the proof of the second conjunct.

To prove that the new history state binding is authenticated in the primed state, we prove that the new history state binding does not change, and we prove that the new history state binding was authenticated in the unprimed state. Since, by assumption of the lemma, the authentication status of any history state binding does not change, the new history state binding is authenticated in the primed state.

$\langle 1 \rangle 6.$ $LL1HistoryStateBindingAuthenticated(newHistoryStateBinding)'$

One fact we'll need several times is that the symmetric key in the $NVRAM$ has not changed, so we'll prove this once up front.

$\langle 2 \rangle 1.$ UNCHANGED $LL1NVRAM.symmetricKey$

$\langle 3 \rangle 1.$ UNCHANGED $LL1NVRAM$

BY $\langle 1 \rangle 1$

$\langle 3 \rangle 2.$ QED

BY $\langle 3 \rangle 1$

In the unprimed state, the new history state binding was authenticated. This follows because the *InclusionInvariant* is true in the unprimed state.

$\langle 2 \rangle 2.$ $LL1HistoryStateBindingAuthenticated(newHistoryStateBinding)$

We prove the two conjuncts in the antecedent of the *InclusionInvariant*. The first conjunct follows as a straightforward consequence of the fact that $NVRAM$ has not changed.

$\langle 3 \rangle 1.$ $LL1NVRAM.historySummary = Hash(historySummary,\ input)$

$\langle 4 \rangle 1.$ $LL1NVRAM.historySummary' = Hash(historySummary,\ input)$

BY $\langle 1 \rangle 4$

$\langle 4 \rangle 2.$ UNCHANGED $LL1NVRAM.historySummary$

$\langle 5 \rangle 1.$ UNCHANGED $LL1NVRAM$

BY $\langle 1 \rangle 1$

$\langle 5 \rangle 2.$ QED

BY $\langle 5 \rangle 1$

$\langle 4 \rangle 3.$ QED

BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$

Proving the second conjuct is more involved. We'll show that the history state binding is authenticated in the primed state and that its authentication status has not changed

$\langle 3 \rangle 2.$ $LL1HistoryStateBindingAuthenticated(historyStateBinding)$

The history state binding is authenticated in the primed state, by hypothesis.

$\langle 4 \rangle 1.$ $LL1HistoryStateBindingAuthenticated(historyStateBinding)'$

BY $\langle 1 \rangle 4$

The authentication status of the history state binding has not changed, because this status has not changed for any history state binding in *HashType*.

$\langle 4 \rangle 2.$ UNCHANGED $LL1HistoryStateBindingAuthenticated(historyStateBinding)$

We have to show that the history state binding has the appropriate type.

$\langle 5 \rangle 1.$ $historyStateBinding \in HashType$

BY $\langle 1 \rangle 3$

The authentication status of all input state bindings has not changed, as assumed by the lemma.

$\langle 5 \rangle 2.$ $\forall\, historyStateBinding1 \in HashType :$
UNCHANGED $LL1HistoryStateBindingAuthenticated(historyStateBinding1)$

BY $\langle 1 \rangle 1$

The conclusion follows directly.

$\langle 5 \rangle 3.$ QED

BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$

Since the history state binding is authenticated in the primed state, and since its authentication status has not changed, it is also authenticated in the unprimed state.

$\langle 4 \rangle 3.$ QED

BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$

We can then use the *InclusionInvariant* to prove that the new history state binding was authenticated in the unprimed state.

⟨3⟩3. *InclusionInvariant*
  BY ⟨1⟩1
⟨3⟩4. QED
  BY ⟨1⟩2, ⟨3⟩1, ⟨3⟩2, ⟨3⟩3
  DEF *InclusionInvariant*, *newHistoryStateBinding*, *newStateHash*, *newPrivateStateEnc*, *sResult*, *privateState*, *historyStateBinding*, *stateHash*
⟨2⟩3. UNCHANGED *LL1HistoryStateBindingAuthenticated*(*newHistoryStateBinding*)

The new history state binding has not changed.

⟨3⟩1. UNCHANGED *newHistoryStateBinding*
  ⟨4⟩1. UNCHANGED *LL1NVRAM.historySummary*
    ⟨5⟩1. UNCHANGED *LL1NVRAM*
      BY ⟨1⟩1
    ⟨5⟩2. QED
      BY ⟨5⟩1
  ⟨4⟩2. UNCHANGED *newStateHash*
    ⟨5⟩1. UNCHANGED *sResult*
      ⟨6⟩1. UNCHANGED *privateState*
        BY ⟨2⟩1 DEF *privateState*
      ⟨6⟩2. QED
        BY ⟨6⟩1 DEF *sResult*
    ⟨5⟩2. UNCHANGED *sResult.newPublicState*
      BY ⟨5⟩1
    ⟨5⟩3. UNCHANGED *newPrivateStateEnc*
      ⟨6⟩1. UNCHANGED *sResult.newPrivateState*
        BY ⟨5⟩1
      ⟨6⟩2. QED
        BY ⟨2⟩1, ⟨6⟩1 DEF *newPrivateStateEnc*
    ⟨5⟩4. QED
      BY ⟨5⟩2, ⟨5⟩3 DEF *newStateHash*
  ⟨4⟩3. QED
    BY ⟨4⟩1, ⟨4⟩2 DEF *newHistoryStateBinding*

The new history state binding has the appropriate type.

⟨3⟩2. *newHistoryStateBinding* ∈ *HashType*
  BY ⟨1⟩3

The authentication status of all input state bindings has not changed, as assumed by the lemma.

⟨3⟩3. ∀ *historyStateBinding1* ∈ *HashType* :
    UNCHANGED *LL1HistoryStateBindingAuthenticated*(*historyStateBinding1*)
  BY ⟨1⟩1
⟨3⟩4. QED
  BY ⟨3⟩1, ⟨3⟩2, ⟨3⟩3
⟨2⟩4. QED
  BY ⟨2⟩2, ⟨2⟩3

Each of the conjuncts is true, so the conjunction is true.

⟨1⟩7. QED
  BY ⟨1⟩5, ⟨1⟩6

The *CardinalityUnchangedLemma* states that, if there are no changes to the *NVRAM* or to the authentication status of any history state binding, then the *CardinalityInvariant* holds inductively from the unprimed state to the primed state.

THEOREM $\mathit{CardinalityUnchangedLemma} \triangleq$

$(\land \; \mathit{LL1TypeInvariant}$
$\;\; \land \; \mathit{CardinalityInvariant}$
$\;\; \land \; \textsc{unchanged } \mathit{LL1NVRAM}$
$\;\; \land \; \forall \mathit{historyStateBinding1} \in \mathit{HashType} :$
$\quad\quad \textsc{unchanged } \mathit{LL1HistoryStateBindingAuthenticated}(\mathit{historyStateBinding1}))$
$\Rightarrow$
$\quad \mathit{CardinalityInvariant}'$

We begin by assuming the antecedent.

⟨1⟩1. HAVE $\land \; \mathit{LL1TypeInvariant}$
$\land \; \mathit{CardinalityInvariant}$
$\land \; \textsc{unchanged } \mathit{LL1NVRAM}$
$\land \; \forall \mathit{historyStateBinding} \in \mathit{HashType} :$
$\quad \textsc{unchanged } \mathit{LL1HistoryStateBindingAuthenticated}(\mathit{historyStateBinding})$

To prove the universally quantified expression, we take a new set of variables in the appropriate types. For the TAKE step to be meaningful to the prover, first we have to tell the prover to expand the definition of *CardinalityInvariant*, so it will see the universally quantified expression therein.

⟨1⟩ USE DEF *CardinalityInvariant*

⟨1⟩2. TAKE $\mathit{historySummary} \in \mathit{HashType}, \; \mathit{stateHash} \in \mathit{HashType}$

To simplify the writing of the proof, we re-state the definition from the *CardinalityInvariant*.

⟨1⟩ $\mathit{historyStateBinding} \triangleq \mathit{Hash}(\mathit{historySummary}, \mathit{stateHash})$

We then assert the type safety of these definitions, with the help of the *CardinalityInvariantDefsTypeSafeLemma*.

⟨1⟩3. $\mathit{historyStateBinding} \in \mathit{HashType}$
BY ⟨1⟩2, *CardinalityInvariantDefsTypeSafeLemma*

The *CardinalityInvariant* states an implication. To prove this, it suffices to assume the antecedent and prove the consequent.

⟨1⟩4. SUFFICES
ASSUME
$\land \; \mathit{LL1NVRAMHistorySummaryUncorrupted}'$
$\land \; \mathit{LL1HistoryStateBindingAuthenticated}(\mathit{historyStateBinding})'$
PROVE
$\mathit{HashCardinality}(\mathit{historySummary}) \leq \mathit{HashCardinality}(\mathit{LL1NVRAM.historySummary}')$
OBVIOUS

We hide the definitions, so they don't overwhelm the prover. We'll pull them in as necessary below.

⟨1⟩ HIDE DEF *CardinalityInvariant*
⟨1⟩ HIDE DEF *historyStateBinding*

To prove the inequality in the primed state, we prove two things: (1) Before the action, the inequality held. (2) The history summary in the *NVRAM* does not change.

The inequality held before the action because the *CardinalityInvariant* was true in the unprimed state.

⟨1⟩5. $\mathit{HashCardinality}(\mathit{historySummary}) \leq \mathit{HashCardinality}(\mathit{LL1NVRAM.historySummary})$

We'll prove the two antecedents of the *CardinalityInvariant* in the unprimed state. First, we'll prove that the *LL1NVRAMHistorySummaryUncorrupted* predicate is true.

⟨2⟩1. *LL1NVRAMHistorySummaryUncorrupted*

The *LL1NVRAMHistorySummaryUncorrupted* predicate is true in the primed state, by hypothesis.

⟨3⟩1. $\mathit{LL1NVRAMHistorySummaryUncorrupted}'$
BY ⟨1⟩4

The *LL1NVRAMHistorySummaryUncorrupted* predicate has not changed, thanks to the *LL1NVRAMHistorySummaryUncorruptedUnchangedLemma*.

⟨3⟩2. UNCHANGED *LL1NVRAMHistorySummaryUncorrupted*
BY ⟨1⟩1, *LL1NVRAMHistorySummaryUncorruptedUnchangedLemma*

Since the *LL1NVRAMHistorySummaryUncorrupted* predicate is true in the primed state, and since the *LL1NVRAMHistorySummaryUncorrupted* predicate has not changed, the predicate is also true in the unprimed state.

$\langle 3 \rangle 3.$ QED
BY $\langle 3 \rangle 1$, $\langle 3 \rangle 2$

Then, we'll prove that the history state binding is authenticated in the unprimed state. This is the second antecedent of the *CardinalityInvariant*.

$\langle 2 \rangle 2.$ $\mathit{LL1HistoryStateBindingAuthenticated}(\mathit{historyStateBinding})$

The history state binding is authenticated in the primed state, by hypothesis.

$\langle 3 \rangle 1.$ $\mathit{LL1HistoryStateBindingAuthenticated}(\mathit{historyStateBinding})'$
BY $\langle 1 \rangle 4$

The authentication status of the history state binding has not changed, because this status has not changed for any history state binding in *HashType*.

$\langle 3 \rangle 2.$ UNCHANGED $\mathit{LL1HistoryStateBindingAuthenticated}(\mathit{historyStateBinding})$

We have to show that the history state binding has the appropriate type.

$\langle 4 \rangle 1.$ $\mathit{historyStateBinding} \in \mathit{HashType}$
BY $\langle 1 \rangle 3$

The authentication status of all input state bindings has not changed, as assumed by the lemma.

$\langle 4 \rangle 2.$ $\forall\, \mathit{historyStateBinding1} \in \mathit{HashType}$:
UNCHANGED $\mathit{LL1HistoryStateBindingAuthenticated}(\mathit{historyStateBinding1})$
BY $\langle 1 \rangle 1$

The conclusion follows directly.

$\langle 4 \rangle 3.$ QED
BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$

Since the history state binding is authenticated in the primed state, and since its authentication status has not changed, it is also authenticated in the unprimed state.

$\langle 3 \rangle 3.$ QED
BY $\langle 3 \rangle 1$, $\langle 3 \rangle 2$

We can then use the *CardinalityInvariant* to prove that the inquality held in the unprimed state.

$\langle 2 \rangle 3.$ *CardinalityInvariant*
BY $\langle 1 \rangle 1$

$\langle 2 \rangle 4.$ QED
BY $\langle 1 \rangle 2$, $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, $\langle 2 \rangle 3$
DEF *CardinalityInvariant*, *historyStateBinding*

Step 2: The history summary in the *NVRAM* does not change.

$\langle 1 \rangle 6.$ UNCHANGED $\mathit{LL1NVRAM.historySummary}$
$\langle 2 \rangle 1.$ UNCHANGED $\mathit{LL1NVRAM}$
BY $\langle 1 \rangle 1$
$\langle 2 \rangle 2.$ QED
BY $\langle 2 \rangle 1$
$\langle 1 \rangle 7.$ QED
BY $\langle 1 \rangle 5$, $\langle 1 \rangle 6$

The *UniquenessUnchangedLemma* states that, if there are no changes to the *NVRAM* or to the authentication status of any history state binding, then the *UniquenessInvariant* holds inductively from the unprimed state to the primed state.

THEOREM *UniquenessUnchangedLemma* $\triangleq$
($\wedge$ *LL1TypeInvariant*
$\wedge$ *UniquenessInvariant*

$\land$ UNCHANGED $LL1NVRAM$
$\land$ $\forall\, historyStateBinding \in HashType$:
UNCHANGED $LL1HistoryStateBindingAuthenticated(historyStateBinding))$
$\Rightarrow$
$UniquenessInvariant'$

We begin by assuming the antecedent.

$\langle 1 \rangle 1.$ HAVE $\land$ $LL1TypeInvariant$
$\land$ $UniquenessInvariant$
$\land$ UNCHANGED $LL1NVRAM$
$\land$ $\forall\, historyStateBinding \in HashType$:
UNCHANGED $LL1HistoryStateBindingAuthenticated(historyStateBinding)$

To prove the universally quantified expression, we take a new set of variables in the appropriate types. For the TAKE step to be meaningful to the prover, first we have to tell the prover to expand the definition of *UniquenessInvariant*, so it will see the universally quantified expression therein.

$\langle 1 \rangle$ USE DEF $UniquenessInvariant$
$\langle 1 \rangle 2.$ TAKE $stateHash1,\ stateHash2 \in HashType$

To simplify the writing of the proof, we re-state the definitions from the *UniquenessInvariant*.

$\langle 1 \rangle$ $historyStateBinding1 \triangleq Hash(LL1NVRAM.historySummary,\ stateHash1)$
$\langle 1 \rangle$ $historyStateBinding2 \triangleq Hash(LL1NVRAM.historySummary,\ stateHash2)$

We then assert the type safety of these definitions, with the help of the *UniquenessInvariantDefsTypeSafeLemma*.

$\langle 1 \rangle 3.$ $\land$ $historyStateBinding1 \in HashType$
$\land$ $historyStateBinding2 \in HashType$
$\langle 2 \rangle 1.$ $LL1TypeInvariant$
BY $\langle 1 \rangle 1$
$\langle 2 \rangle 2.$ QED
BY $\langle 1 \rangle 2$, $\langle 2 \rangle 1$, $UniquenessInvariantDefsTypeSafeLemma$

The *UniquenessInvariant* states an implication. To prove this, it suffices to assume the antecedent and prove the consequent.

$\langle 1 \rangle 4.$ SUFFICES
ASSUME $\land$ $LL1HistoryStateBindingAuthenticated(historyStateBinding1)'$
$\land$ $LL1HistoryStateBindingAuthenticated(historyStateBinding2)'$
PROVE $stateHash1 = stateHash2$
OBVIOUS

We hide the definitions, so they don't overwhelm the prover. We'll pull them in as necessary below.

$\langle 1 \rangle$ HIDE DEF $UniquenessInvariant$
$\langle 1 \rangle$ HIDE DEF $historyStateBinding1,\ historyStateBinding2$

First we'll show that history state binding 1 is authenticated in the unprimed state.

$\langle 1 \rangle 5.$ $LL1HistoryStateBindingAuthenticated(historyStateBinding1)$

History state binding 1 is authenticated in the primed state, by hypothesis.

$\langle 2 \rangle 1.$ $LL1HistoryStateBindingAuthenticated(historyStateBinding1)'$
BY $\langle 1 \rangle 4$

The authentication status of history state binding 1 has not changed, because this status has not changed for any history state binding in *HashType*.

$\langle 2 \rangle 2.$ UNCHANGED $LL1HistoryStateBindingAuthenticated(historyStateBinding1)$

We have to show that the history state binding has not changed, which is not obvious because it is defined in terms of the history summary in the *NVRAM*, so we have to derive this from the fact that the *NVRAM* has not changed.

$\langle 3 \rangle 1.$ UNCHANGED $historyStateBinding1$
$\langle 4 \rangle 1.$ UNCHANGED $LL1NVRAM.historySummary$
$\langle 5 \rangle 1.$ UNCHANGED $LL1NVRAM$
BY $\langle 1 \rangle 1$

$\langle 5 \rangle 2.$ QED

BY $\langle 5 \rangle 1$

$\langle 4 \rangle 2.$ QED

BY $\langle 4 \rangle 1$ DEF *historyStateBinding*1

Then, we have to show that history state binding 1 has the appropriate type.

$\langle 3 \rangle 2.$ $historyStateBinding1 \in HashType$

BY $\langle 1 \rangle 3$

The authentication status of all input state bindings has not changed, as assumed by the lemma.

$\langle 3 \rangle 3.$ $\forall\, historyStateBinding \in HashType :$

UNCHANGED $LL1HistoryStateBindingAuthenticated(historyStateBinding)$

BY $\langle 1 \rangle 1$

The conclusion follows directly.

$\langle 3 \rangle 4.$ QED

BY $\langle 3 \rangle 1$, $\langle 3 \rangle 2$, $\langle 3 \rangle 3$

Since history state binding 1 is authenticated in the primed state, and since its authentication status has not changed, it is also authenticated in the unprimed state.

$\langle 2 \rangle 3.$ QED

BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$

The same argument holds for history state binding 2.

$\langle 1 \rangle 6.$ $LL1HistoryStateBindingAuthenticated(historyStateBinding2)$

History state binding 2 is authenticated in the primed state, by hypothesis.

$\langle 2 \rangle 1.$ $LL1HistoryStateBindingAuthenticated(historyStateBinding2)'$

BY $\langle 1 \rangle 4$

The authentication status of history state binding 2 has not changed, because this status has not changed for any history state binding in *HashType*.

$\langle 2 \rangle 2.$ UNCHANGED $LL1HistoryStateBindingAuthenticated(historyStateBinding2)$

We have to show that the history state binding has not changed, which is not obvious because it is defined in terms of the history summary in the *NVRAM*, so we have to derive this from the fact that the *NVRAM* has not changed.

$\langle 3 \rangle 1.$ UNCHANGED *historyStateBinding*2

$\langle 4 \rangle 1.$ UNCHANGED $LL1NVRAM.historySummary$

$\langle 5 \rangle 1.$ UNCHANGED $LL1NVRAM$

BY $\langle 1 \rangle 1$

$\langle 5 \rangle 2.$ QED

BY $\langle 5 \rangle 1$

$\langle 4 \rangle 2.$ QED

BY $\langle 4 \rangle 1$ DEF *historyStateBinding*2

Then, we have to show that history state binding 2 has the appropriate type.

$\langle 3 \rangle 2.$ $historyStateBinding2 \in HashType$

BY $\langle 1 \rangle 3$

The authentication status of all input state bindings has not changed, as assumed by the lemma.

$\langle 3 \rangle 3.$ $\forall\, historyStateBinding \in HashType :$

UNCHANGED $LL1HistoryStateBindingAuthenticated(historyStateBinding)$

BY $\langle 1 \rangle 1$

The conclusion follows directly.

$\langle 3 \rangle 4.$ QED

BY $\langle 3 \rangle 1$, $\langle 3 \rangle 2$, $\langle 3 \rangle 3$

Since history state binding 1 is authenticated in the primed state, and since its authentication status has not changed, it is also authenticated in the unprimed state.

$\langle 2 \rangle 3.$ QED

BY ⟨2⟩1, ⟨2⟩2

Since the *UniquenessInvariant* holds in the unprimed state, it follows directly that the two state hashes are equal.

⟨1⟩7. QED

⟨2⟩1. $stateHash1 \in HashType$

BY ⟨1⟩2

⟨2⟩2. $stateHash2 \in HashType$

BY ⟨1⟩2

⟨2⟩3. *UniquenessInvariant*

BY ⟨1⟩1

⟨2⟩4. QED

BY ⟨1⟩5, ⟨1⟩6, ⟨2⟩1, ⟨2⟩2, ⟨2⟩3

DEF *UniquenessInvariant*, *historyStateBinding1*, *historyStateBinding2*

The *UnchangedAuthenticatedHistoryStateBindingsLemma* states that if the *NVRAM* and the set of observed authenticators does not change, then there is no change to the set of history state bindings that have authenticators in the set *LL1ObservedAuthenticators*.

THEOREM *UnchangedAuthenticatedHistoryStateBindingsLemma* $\triangleq$

UNCHANGED $\langle LL1NVRAM, LL1ObservedAuthenticators \rangle \Rightarrow$

$\forall historyStateBinding \in HashType :$

UNCHANGED $LL1HistoryStateBindingAuthenticated(historyStateBinding)$

We assume the antecedents.

⟨1⟩1. HAVE UNCHANGED $\langle LL1NVRAM, LL1ObservedAuthenticators \rangle$

To prove the universally quantified expression, we take a new history state binding in *HashType*.

⟨1⟩2. TAKE $historyStateBinding \in HashType$

One fact that will be useful in several places is that the symmetric key in the *NVRAM* has not changed.

⟨1⟩3. UNCHANGED $LL1NVRAM.symmetricKey$

⟨2⟩1. UNCHANGED $LL1NVRAM$

BY ⟨1⟩1

⟨2⟩2. QED

BY ⟨2⟩1

We'll subdivide these into two cases. In the first case, we'll consider the history state bindings that are authenticated in the unprimed state, and we'll show that they continue to be authenticated in the primed state.

⟨1⟩4. CASE $LL1HistoryStateBindingAuthenticated(historyStateBinding) =$ TRUE

⟨2⟩1. $LL1HistoryStateBindingAuthenticated(historyStateBinding)' =$ TRUE

By hypothesis, the history state binding is authenticated in the unprimed state. Thus, we can pick an authenticator in the set of observed authenticators that is a valid *MAC* for this history state binding.

⟨3⟩1. PICK $authenticator \in LL1ObservedAuthenticators :$

$ValidateMAC(LL1NVRAM.symmetricKey, historyStateBinding, authenticator)$

BY ⟨1⟩4 DEF *LL1HistoryStateBindingAuthenticated*

Because the symmetric key in the *NVRAM* has not changed, this authenticator is also a valid *MAC* for this history state binding in the primed state.

⟨3⟩2. $ValidateMAC(LL1NVRAM.symmetricKey', historyStateBinding, authenticator)$

BY ⟨1⟩3, ⟨3⟩1

Because the set of observed authenticators has not changed, this authenticator is also in the primed set of observed authenticators.

⟨3⟩3. $authenticator \in LL1ObservedAuthenticators'$

⟨4⟩1. $authenticator \in LL1ObservedAuthenticators$

BY ⟨3⟩1

⟨4⟩2. UNCHANGED $LL1ObservedAuthenticators$

BY ⟨1⟩1

⟨4⟩3. QED

BY ⟨4⟩1, ⟨4⟩2

The previous two conditions are sufficient to establish that the history state binding is authenticated in the primed state.

⟨3⟩4. QED

BY ⟨3⟩2, ⟨3⟩3 DEF *LL1HistoryStateBindingAuthenticated*

Because the history state binding is authenticated in both the unprimed and primed states, the *LL1HistoryStateBindingAuthenticated* is unchanged for this history state binding.

⟨2⟩2. QED

BY ⟨1⟩4, ⟨2⟩1

In the second case, we'll consider the history state bindings that are unauthenticated in the unprimed state, and we'll show that they continue to be unauthenticated in the primed state.

⟨1⟩5. CASE *LL1HistoryStateBindingAuthenticated*(*historyStateBinding*) = FALSE

⟨2⟩1. *LL1HistoryStateBindingAuthenticated*(*historyStateBinding*)′ = FALSE

To prove that the history state binding is not authenticated in the primed state, it suffices to show that none of the state authenticators in the primed set of observed authenticators is a valid MAC for the history state binding.

⟨3⟩1. SUFFICES $\forall$ *authenticator* $\in$ $LL1ObservedAuthenticators'$:

$\neg$ *ValidateMAC*($LL1NVRAM.symmetricKey'$, *historyStateBinding*, *authenticator*)

BY DEF *LL1HistoryStateBindingAuthenticated*

To prove the universally quantified expression, we take a new authenticator in the primed set of observed authenticators.

⟨3⟩2. TAKE *authenticator* $\in$ $LL1ObservedAuthenticators'$

Because the set of observed authenticators has not changed, this authenticator is also in the unprimed set of observed authenticators.

⟨3⟩3. *authenticator* $\in$ $LL1ObservedAuthenticators$

⟨4⟩1. UNCHANGED $LL1ObservedAuthenticators$

BY ⟨1⟩1

⟨4⟩2. QED

BY ⟨3⟩2, ⟨4⟩1

Because the authenticator failed to authenticate the history state binding in the unprimed state, and the symmetric key has not changed, it immediately follows that the authenticator will not authenticate the history state binding in the primed state.

⟨3⟩4. QED

BY ⟨1⟩3, ⟨1⟩5, ⟨3⟩3 DEF *LL1HistoryStateBindingAuthenticated*

Because the history state binding is unauthenticated in both the unprimed and primed states, the *LL1HistoryStateBindingAuthenticated* is unchanged for this history state binding.

⟨2⟩2. QED

BY ⟨1⟩5, ⟨2⟩1

By proving that *LL1HistoryStateBindingAuthenticated* is a boolean predicate, it is immediately clear that the two cases of true and false are exhaustive for this predicate.

⟨1⟩6. *LL1HistoryStateBindingAuthenticated*(*historyStateBinding*) $\in$ BOOLEAN

BY DEF *LL1HistoryStateBindingAuthenticated*

⟨1⟩7. QED

BY ⟨1⟩4, ⟨1⟩5, ⟨1⟩6

4.5 Proof of Unforgeability Invariance in Memoir-Basic

MODULE *MemoirLL1UnforgeabilityInvariance*

This module proves that the *UnforgeabilityInvariant* is an inductive invariant of the Memoir-Basic spec.

EXTENDS *MemoirLL1InvarianceLemmas*

Because the spec allows the data on the disk to be read into the RAM, proving *UnforgeabilityInvariance* of the RAM requires also proving an analagous property for the disk. Thus, we first prove the that the *ExtendedUnforgeabilityInvariant* is an invariant of the spec.

THEOREM $\mathit{ExtendedUnforgeabilityInvariance} \triangleq \mathit{LL1Spec} \Rightarrow \Box \mathit{ExtendedUnforgeabilityInvariant}$

This proof will require the *LL1TypeInvariant*. Fortunately, the *LL1TypeSafe* theorem has already proven that the Memoir-Basic spec satisfies its type invariant.

$\langle 1\rangle 1.$ $\mathit{LL1Spec} \Rightarrow \Box \mathit{LL1TypeInvariant}$
  BY *LL1TypeSafe*

The top level of the proof is boilerplate TLA+ for an *Inv*1-style proof. First, we prove that the initial state satisfies *ExtendedUnforgeabilityInvariant*. Second, we prove that the *LL1Next* predicate inductively preserves *ExtendedUnforgeabilityInvariant*. Third, we use temporal induction to prove that these two conditions satisfy the *ExtendedUnforgeabilityInvariant* over all behaviors.

$\langle 1\rangle 2.$ $\mathit{LL1Init} \land \mathit{LL1TypeInvariant} \Rightarrow \mathit{ExtendedUnforgeabilityInvariant}$

First, we assume the antecedents.

$\langle 2\rangle 1.$ HAVE $\mathit{LL1Init} \land \mathit{LL1TypeInvariant}$

Then, we pick some symmetrick key for which *LL1Init* is true.

$\langle 2\rangle 2.$ PICK $\mathit{symmetricKey} \in \mathit{SymmetricKeyType} : \mathit{LL1Init}!(\mathit{symmetricKey})!1$
  BY $\langle 2\rangle 1$ DEF *LL1Init*

To simplify the writing of the proof, we re-state some of the definitions from *LL1Init*. We don't need all of them for this proof, so we only re-state the ones we need.

$\langle 2\rangle$ $\mathit{initialPrivateStateEnc} \triangleq \mathit{SymmetricEncrypt}(\mathit{symmetricKey}, \mathit{InitialPrivateState})$
$\langle 2\rangle$ $\mathit{initialStateHash} \triangleq \mathit{Hash}(\mathit{InitialPublicState}, \mathit{initialPrivateStateEnc})$
$\langle 2\rangle$ $\mathit{initialHistoryStateBinding} \triangleq \mathit{Hash}(\mathit{BaseHashValue}, \mathit{initialStateHash})$
$\langle 2\rangle$ $\mathit{initialAuthenticator} \triangleq \mathit{GenerateMAC}(\mathit{symmetricKey}, \mathit{initialHistoryStateBinding})$

We hide the definitions, so they don't overwhelm the prover. We'll pull them in as necessary below.

$\langle 2\rangle$ HIDE DEF *initialPrivateStateEnc*, *initialStateHash*, *initialHistoryStateBinding*, *initialAuthenticator*

To prove the universally quantified expression, we take a new hash. For the TAKE step to be meaningful to the prover, first we have to tell the prover to expand the definition of *ExtendedUnforgeabilityInvariant*, so it will see the universally quantified expression therein.

$\langle 2\rangle$ USE DEF *ExtendedUnforgeabilityInvariant*
$\langle 2\rangle 3.$ TAKE $\mathit{historyStateBinding} \in \mathit{HashType}$

We will prove each of the conjuncts separately. Following is the proof of the unforgeability invariant for the RAM. It follows directly from the definition of *LL1Init*.

$\langle 2\rangle 4.$ $\mathit{ValidateMAC}(\mathit{LL1NVRAM}.\mathit{symmetricKey}, \mathit{historyStateBinding}, \mathit{LL1RAM}.\mathit{authenticator}) \Rightarrow \mathit{LL1RAM}.\mathit{authenticator} \in \mathit{LL1ObservedAuthenticators}$
  $\langle 3\rangle 1.$ HAVE $\mathit{ValidateMAC}(\mathit{LL1NVRAM}.\mathit{symmetricKey}, \mathit{historyStateBinding}, \mathit{LL1RAM}.\mathit{authenticator})$
  $\langle 3\rangle 2.$ $\mathit{LL1RAM}.\mathit{authenticator} = \mathit{initialAuthenticator}$
    BY $\langle 2\rangle 2$ DEF *initialAuthenticator*, *initialHistoryStateBinding*, *initialStateHash*, *initialPrivateStateEnc*
  $\langle 3\rangle 3.$ $\mathit{LL1ObservedAuthenticators} = \{\mathit{initialAuthenticator}\}$
    BY $\langle 2\rangle 2$ DEF *initialAuthenticator*, *initialHistoryStateBinding*, *initialStateHash*, *initialPrivateStateEnc*
  $\langle 3\rangle 4.$ QED

BY $\langle 3\rangle 2$, $\langle 3\rangle 3$

Following is the proof of the unforgeability invariant for the disk. It follows directly from the definition of *LL1Init*.

$\langle 2\rangle 5$. $ValidateMAC(LL1NVRAM.symmetricKey, historyStateBinding, LL1Disk.authenticator) \Rightarrow$
$LL1Disk.authenticator \in LL1ObservedAuthenticators$

$\langle 3\rangle 1$. HAVE $ValidateMAC(LL1NVRAM.symmetricKey, historyStateBinding, LL1Disk.authenticator)$

$\langle 3\rangle 2$. $LL1Disk.authenticator = initialAuthenticator$

BY $\langle 2\rangle 2$ DEF $initialAuthenticator$, $initialHistoryStateBinding$, $initialStateHash$, $initialPrivateStateEnc$

$\langle 3\rangle 3$. $LL1ObservedAuthenticators = \{initialAuthenticator\}$

BY $\langle 2\rangle 2$ DEF $initialAuthenticator$, $initialHistoryStateBinding$, $initialStateHash$, $initialPrivateStateEnc$

$\langle 3\rangle 4$. QED

BY $\langle 3\rangle 2$, $\langle 3\rangle 3$

$\langle 2\rangle 6$. QED

BY $\langle 2\rangle 4$, $\langle 2\rangle 5$

For the induction step, we will need the type invariant to be true in both the unprimed and primed states.

$\langle 1\rangle 3$. $(\land\ ExtendedUnforgeabilityInvariant$
$\land\ [LL1Next]_{LL1Vars}$
$\land\ LL1TypeInvariant$
$\land\ LL1TypeInvariant')$
$\Rightarrow$
$ExtendedUnforgeabilityInvariant'$

First, we assume the antecedents.

$\langle 2\rangle 1$. HAVE $ExtendedUnforgeabilityInvariant \land [LL1Next]_{LL1Vars} \land LL1TypeInvariant \land LL1TypeInvariant'$

The induction step includes two cases: stuttering and *LL1Next* actions. The stuttering case is a straightforward application of the *LL1RAMUnforgeabilityUnchangedLemma* and the *LL1DiskUnforgeabilityUnchangedLemma*.

$\langle 2\rangle 2$. CASE UNCHANGED $LL1Vars$

$\langle 3\rangle 1$. $\forall\, historyStateBinding \in HashType :$
$ValidateMAC(LL1NVRAM.symmetricKey', historyStateBinding, LL1RAM.authenticator') \Rightarrow$
$LL1RAM.authenticator' \in LL1ObservedAuthenticators'$

$\langle 4\rangle 1$. $ExtendedUnforgeabilityInvariant \land LL1TypeInvariant \land LL1TypeInvariant'$

BY $\langle 2\rangle 1$

$\langle 4\rangle 2$. UNCHANGED $LL1RAM$

BY $\langle 2\rangle 2$ DEF $LL1Vars$

$\langle 4\rangle 3$. UNCHANGED $LL1ObservedAuthenticators$

BY $\langle 2\rangle 2$ DEF $LL1Vars$

$\langle 4\rangle 4$. UNCHANGED $LL1NVRAM.symmetricKey$

$\langle 5\rangle 1$. UNCHANGED $LL1NVRAM$

BY $\langle 2\rangle 2$ DEF $LL1Vars$

$\langle 5\rangle 2$. QED

BY $\langle 5\rangle 1$

$\langle 4\rangle 5$. QED

BY $\langle 4\rangle 1$, $\langle 4\rangle 2$, $\langle 4\rangle 3$, $\langle 4\rangle 4$, $LL1RAMUnforgeabilityUnchangedLemma$

$\langle 3\rangle 2$. $\forall\, historyStateBinding \in HashType :$
$ValidateMAC(LL1NVRAM.symmetricKey', historyStateBinding, LL1Disk.authenticator') \Rightarrow$
$LL1Disk.authenticator' \in LL1ObservedAuthenticators'$

$\langle 4\rangle 1$. $ExtendedUnforgeabilityInvariant \land LL1TypeInvariant \land LL1TypeInvariant'$

BY $\langle 2\rangle 1$

$\langle 4\rangle 2$. UNCHANGED $LL1Disk$

BY $\langle 2\rangle 2$ DEF $LL1Vars$

$\langle 4\rangle 3$. UNCHANGED $LL1ObservedAuthenticators$

BY $\langle 2\rangle 2$ DEF *LL1Vars*

$\langle 4\rangle 4$. UNCHANGED *LL1NVRAM.symmetricKey*

$\langle 5\rangle 1$. UNCHANGED *LL1NVRAM*

BY $\langle 2\rangle 2$ DEF *LL1Vars*

$\langle 5\rangle 2$. QED

BY $\langle 5\rangle 1$

$\langle 4\rangle 5$. QED

BY $\langle 4\rangle 1$, $\langle 4\rangle 2$, $\langle 4\rangle 3$, $\langle 4\rangle 4$, *LL1DiskUnforgeabilityUnchangedLemma*

$\langle 3\rangle 3$. QED

BY $\langle 3\rangle 1$, $\langle 3\rangle 2$ DEF *ExtendedUnforgeabilityInvariant*

We break down the *LL1Next* case into eight separate cases for each action.

$\langle 2\rangle 3$. CASE *LL1Next*

The *LL1MakeInputAvailable* case is a straightforward application of the *LL1RAMUnforgeabilityUnchangedLemma* and the *LL1DiskUnforgeabilityUnchangedLemma*.

$\langle 3\rangle 1$. CASE *LL1MakeInputAvailable*

$\langle 4\rangle 1$. PICK $input \in InputType : LL1MakeInputAvailable!(input)$

BY $\langle 3\rangle 1$ DEF *LL1MakeInputAvailable*

$\langle 4\rangle 2$. $\forall\, historyStateBinding \in HashType :$
$$ValidateMAC(LL1NVRAM.symmetricKey', historyStateBinding, LL1RAM.authenticator') \Rightarrow LL1RAM.authenticator' \in LL1ObservedAuthenticators'$$

$\langle 5\rangle 1$. $ExtendedUnforgeabilityInvariant \land LL1TypeInvariant \land LL1TypeInvariant'$

BY $\langle 2\rangle 1$

$\langle 5\rangle 2$. UNCHANGED *LL1RAM*

BY $\langle 4\rangle 1$

$\langle 5\rangle 3$. UNCHANGED *LL1ObservedAuthenticators*

BY $\langle 4\rangle 1$

$\langle 5\rangle 4$. UNCHANGED *LL1NVRAM.symmetricKey*

$\langle 6\rangle 1$. UNCHANGED *LL1NVRAM*

BY $\langle 4\rangle 1$

$\langle 6\rangle 2$. QED

BY $\langle 6\rangle 1$

$\langle 5\rangle 5$. QED

BY $\langle 5\rangle 1$, $\langle 5\rangle 2$, $\langle 5\rangle 3$, $\langle 5\rangle 4$, *LL1RAMUnforgeabilityUnchangedLemma*

$\langle 4\rangle 3$. $\forall\, historyStateBinding \in HashType :$
$$ValidateMAC(LL1NVRAM.symmetricKey', historyStateBinding, LL1Disk.authenticator') \Rightarrow LL1Disk.authenticator' \in LL1ObservedAuthenticators'$$

$\langle 5\rangle 1$. $ExtendedUnforgeabilityInvariant \land LL1TypeInvariant \land LL1TypeInvariant'$

BY $\langle 2\rangle 1$

$\langle 5\rangle 2$. UNCHANGED *LL1Disk*

BY $\langle 4\rangle 1$

$\langle 5\rangle 3$. UNCHANGED *LL1ObservedAuthenticators*

BY $\langle 4\rangle 1$

$\langle 5\rangle 4$. UNCHANGED *LL1NVRAM.symmetricKey*

$\langle 6\rangle 1$. UNCHANGED *LL1NVRAM*

BY $\langle 4\rangle 1$

$\langle 6\rangle 2$. QED

BY $\langle 6\rangle 1$

$\langle 5\rangle 5$. QED

BY $\langle 5\rangle 1$, $\langle 5\rangle 2$, $\langle 5\rangle 3$, $\langle 5\rangle 4$, *LL1DiskUnforgeabilityUnchangedLemma*

$\langle 4\rangle 4$. QED

BY $\langle 4\rangle 2$, $\langle 4\rangle 3$ DEF *ExtendedUnforgeabilityInvariant*

The *LL1PerformOperation* case is not terribly involved, but we have to treat the RAM and disk conjuncts separately.

$\langle 3\rangle 2.$ CASE $LL1PerformOperation$

We pick some input for which $LL1PerformOperation$ is true.

$\langle 4\rangle 1.$ PICK $input \in LL1AvailableInputs : LL1PerformOperation!(input)!1$
BY $\langle 3\rangle 2$ DEF $LL1PerformOperation$

To simplify the writing of the proof, we re-state the definitions from the $LL1PerformOperation$ action.

$\langle 4\rangle$ $stateHash \triangleq Hash(LL1RAM.publicState, LL1RAM.privateStateEnc)$
$\langle 4\rangle$ $historyStateBinding \triangleq Hash(LL1NVRAM.historySummary, stateHash)$
$\langle 4\rangle$ $privateState \triangleq SymmetricDecrypt(LL1NVRAM.symmetricKey, LL1RAM.privateStateEnc)$
$\langle 4\rangle$ $sResult \triangleq Service(LL1RAM.publicState, privateState, input)$
$\langle 4\rangle$ $newPrivateStateEnc \triangleq$
$SymmetricEncrypt(LL1NVRAM.symmetricKey, sResult.newPrivateState)$
$\langle 4\rangle$ $newHistorySummary \triangleq Hash(LL1NVRAM.historySummary, input)$
$\langle 4\rangle$ $newStateHash \triangleq Hash(sResult.newPublicState, newPrivateStateEnc)$
$\langle 4\rangle$ $newHistoryStateBinding \triangleq Hash(newHistorySummary, newStateHash)$
$\langle 4\rangle$ $newAuthenticator \triangleq GenerateMAC(LL1NVRAM.symmetricKey, newHistoryStateBinding)$

We hide the definitions, so they don't overwhelm the prover. We'll pull them in as necessary below.

$\langle 4\rangle$ HIDE DEF $stateHash$, $historyStateBinding$, $privateState$, $sResult$, $newPrivateStateEnc$,
$newHistorySummary$, $newStateHash$, $newHistoryStateBinding$, $newAuthenticator$

To prove the universally quantified expression, we take a new hash. For the TAKE step to be meaningful to the prover, first we have to tell the prover to expand the definition of $ExtendedUnforgeabilityInvariant$, so it will see the universally quantified expression therein.

$\langle 4\rangle$ USE DEF $ExtendedUnforgeabilityInvariant$
$\langle 4\rangle 2.$ TAKE $historyStateBinding1 \in HashType$

Before proceeding to prove each of the conjuncts, we prove a statement that will be useful in both of the sub-proofs below. Namely, the new authenticator generated by the $LL1PerformOperation$ action is unioned into the set of observed authenticators.

$\langle 4\rangle 3.$ $LL1ObservedAuthenticators' =$
$LL1ObservedAuthenticators \cup \{newAuthenticator\}$
BY $\langle 4\rangle 1$ DEF $newAuthenticator$, $newHistoryStateBinding$, $newStateHash$,
$newHistorySummary$, $newPrivateStateEnc$, $sResult$, $privateState$

For the RAM portion of the unforgeability invariant, we note that the $LL1PerformOperation$ action updates the authenticator in the RAM with the new authenticator. Since this new authenticator is unioned into the set of observed authenticators, the invariant holds in the primed state.

$\langle 4\rangle 4.$ $ValidateMAC(LL1NVRAM.symmetricKey', historyStateBinding1, LL1RAM.authenticator') \Rightarrow$
$LL1RAM.authenticator' \in LL1ObservedAuthenticators'$
$\langle 5\rangle 1.$ $LL1RAM.authenticator' = newAuthenticator$
$\langle 6\rangle 1.$ $newAuthenticator = LL1PerformOperation!(input)!newAuthenticator$
BY $\langle 4\rangle 1$ DEF $newAuthenticator$, $newHistoryStateBinding$, $newStateHash$,
$newHistorySummary$, $newPrivateStateEnc$, $sResult$, $privateState$
$\langle 6\rangle 2.$ $LL1RAM' = [$
$publicState \mapsto LL1PerformOperation!(input)!sResult.newPublicState,$
$privateStateEnc \mapsto LL1PerformOperation!(input)!newPrivateStateEnc,$
$historySummary \mapsto LL1PerformOperation!(input)!newHistorySummary,$
$authenticator \mapsto newAuthenticator]$
BY $\langle 4\rangle 1$, $\langle 6\rangle 1$
$\langle 6\rangle 3.$ QED
BY $\langle 6\rangle 2$
$\langle 5\rangle 2.$ QED
BY $\langle 4\rangle 3$, $\langle 5\rangle 1$

For the disk portion of the unforgeability invariant, we employ the $LL1DiskUnforgeabilityUnchangedLemma$, since the disk is not changed by the $LL1PerformOperation$ action.

$\langle 4 \rangle 5.$ $ValidateMAC(LL1NVRAM.symmetricKey', historyStateBinding1, LL1Disk.authenticator') \Rightarrow$
$LL1Disk.authenticator' \in LL1ObservedAuthenticators'$

$\langle 5 \rangle 1.$ $ExtendedUnforgeabilityInvariant \land LL1TypeInvariant \land LL1TypeInvariant'$
BY $\langle 2 \rangle 1$

$\langle 5 \rangle 2.$ UNCHANGED $LL1Disk$
BY $\langle 4 \rangle 1$

$\langle 5 \rangle 3.$ $LL1ObservedAuthenticators \subseteq LL1ObservedAuthenticators'$
BY $\langle 4 \rangle 3$

$\langle 5 \rangle 4.$ UNCHANGED $LL1NVRAM.symmetricKey$

$\langle 6 \rangle 1.$ $LL1NVRAM' = [historySummary \mapsto LL1PerformOperation!(input)!newHistorySummary,$
$symmetricKey \mapsto LL1NVRAM.symmetricKey]$
BY $\langle 4 \rangle 1$

$\langle 6 \rangle 2.$ $LL1NVRAM.symmetricKey' = LL1NVRAM.symmetricKey$
BY $\langle 6 \rangle 1$

$\langle 6 \rangle 3.$ QED
BY $\langle 6 \rangle 2$

$\langle 5 \rangle 5.$ QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$, $\langle 5 \rangle 3$, $\langle 5 \rangle 4$, $LL1DiskUnforgeabilityUnchangedLemma$

$\langle 4 \rangle 6.$ QED
BY $\langle 4 \rangle 4$, $\langle 4 \rangle 5$

The *LL1RepeatOperation* case is not terribly involved, but we have to treat the RAM and disk conjuncts separately.

$\langle 3 \rangle 3.$ CASE $LL1RepeatOperation$

We pick some input for which *LL1RepeatOperation* is true.

$\langle 4 \rangle 1.$ PICK $input \in LL1AvailableInputs : LL1RepeatOperation!(input)!1$
BY $\langle 3 \rangle 3$ DEF $LL1RepeatOperation$

To simplify the writing of the proof, we re-state the definitions from the *LL1RepeatOperation* action.

$\langle 4 \rangle$ $stateHash \triangleq Hash(LL1RAM.publicState, LL1RAM.privateStateEnc)$
$\langle 4 \rangle$ $historyStateBinding \triangleq Hash(LL1RAM.historySummary, stateHash)$
$\langle 4 \rangle$ $privateState \triangleq SymmetricDecrypt(LL1NVRAM.symmetricKey, LL1RAM.privateStateEnc)$
$\langle 4 \rangle$ $sResult \triangleq Service(LL1RAM.publicState, privateState, input)$
$\langle 4 \rangle$ $newPrivateStateEnc \triangleq$
$SymmetricEncrypt(LL1NVRAM.symmetricKey, sResult.newPrivateState)$
$\langle 4 \rangle$ $newStateHash \triangleq Hash(sResult.newPublicState, newPrivateStateEnc)$
$\langle 4 \rangle$ $newHistoryStateBinding \triangleq Hash(LL1NVRAM.historySummary, newStateHash)$
$\langle 4 \rangle$ $newAuthenticator \triangleq GenerateMAC(LL1NVRAM.symmetricKey, newHistoryStateBinding)$

We hide the definitions, so they don't overwhelm the prover. We'll pull them in as necessary below.

$\langle 4 \rangle$ HIDE DEF $stateHash$, $historyStateBinding$, $privateState$, $sResult$, $newPrivateStateEnc$,
$newStateHash$, $newHistoryStateBinding$, $newAuthenticator$

To prove the universally quantified expression, we take a new hash. For the TAKE step to be meaningful to the prover, first we have to tell the prover to expand the definition of *ExtendedUnforgeabilityInvariant*, so it will see the universally quantified expression therein.

$\langle 4 \rangle$ USE DEF $ExtendedUnforgeabilityInvariant$
$\langle 4 \rangle 2.$ TAKE $historyStateBinding1 \in HashType$

Before proceeding to prove each of the conjuncts, we prove a statement that will be useful in both of the sub-proofs below. Namely, the new authenticator generated by the *LL1PerformOperation* action is unioned into the set of observed authenticators.

$\langle 4 \rangle 3.$ $LL1ObservedAuthenticators' =$
$LL1ObservedAuthenticators \cup \{newAuthenticator\}$
BY $\langle 4 \rangle 1$ DEF $newAuthenticator$, $newHistoryStateBinding$, $newStateHash$,
$newPrivateStateEnc$, $sResult$, $privateState$

For the RAM portion of the unforgeability invariant, we note that the *LL1RepeatOperation* action updates the authenticator in the RAM with the new authenticator. Since this new authenticator is unioned into the set of observed authenticators, the invariant holds in the primed state.

$\langle 4\rangle 4.$ $ValidateMAC(LL1NVRAM.symmetricKey', historyStateBinding1, LL1RAM.authenticator') \Rightarrow$
$\quad LL1RAM.authenticator' \in LL1ObservedAuthenticators'$

$\langle 5\rangle 1.$ $LL1RAM.authenticator' = newAuthenticator$

$\langle 6\rangle 1.$ $newAuthenticator = LL1RepeatOperation!(input)!newAuthenticator$

BY $\langle 4\rangle 1$ DEF $newAuthenticator$, $newHistoryStateBinding$, $newStateHash$, $newPrivateStateEnc$, $sResult$, $privateState$

$\langle 6\rangle 2.$ $LL1RAM' = [$
$publicState \mapsto LL1RepeatOperation!(input)!sResult.newPublicState,$
$privateStateEnc \mapsto LL1RepeatOperation!(input)!newPrivateStateEnc,$
$historySummary \mapsto LL1NVRAM.historySummary,$
$authenticator \mapsto newAuthenticator]$

BY $\langle 4\rangle 1$, $\langle 6\rangle 1$

$\langle 6\rangle 3.$ QED

BY $\langle 6\rangle 2$

$\langle 5\rangle 2.$ QED

BY $\langle 4\rangle 3$, $\langle 5\rangle 1$

For the disk portion of the unforgeability invariant, we employ the *LL1DiskUnforgeabilityUnchangedLemma*, since the disk is not changed by the *LL1RepeatOperation* action.

$\langle 4\rangle 5.$ $ValidateMAC(LL1NVRAM.symmetricKey', historyStateBinding1, LL1Disk.authenticator') \Rightarrow$
$\quad LL1Disk.authenticator' \in LL1ObservedAuthenticators'$

$\langle 5\rangle 1.$ $ExtendedUnforgeabilityInvariant \land LL1TypeInvariant \land LL1TypeInvariant'$

BY $\langle 2\rangle 1$

$\langle 5\rangle 2.$ UNCHANGED $LL1Disk$

BY $\langle 4\rangle 1$

$\langle 5\rangle 3.$ $LL1ObservedAuthenticators \subseteq LL1ObservedAuthenticators'$

BY $\langle 4\rangle 3$

$\langle 5\rangle 4.$ UNCHANGED $LL1NVRAM.symmetricKey$

$\langle 6\rangle 1.$ UNCHANGED $LL1NVRAM$

BY $\langle 4\rangle 1$

$\langle 6\rangle 2.$ QED

BY $\langle 6\rangle 1$

$\langle 5\rangle 5.$ QED

BY $\langle 5\rangle 1$, $\langle 5\rangle 2$, $\langle 5\rangle 3$, $\langle 5\rangle 4$, $LL1RAMUnforgeabilityUnchangedLemma$

$\langle 4\rangle 6.$ QED

BY $\langle 4\rangle 4$, $\langle 4\rangle 5$

The *LL1Restart* case is moderately interesting. The RAM portion of the unforgeability invariant is where we pull in the constraint on the set of authenticators that can wind up in the randomized RAM after a *LL1Restart*. This constraint is explicitly expressed in the definition of the *LL1Restart* action. The disk portion is a straightforward application of the *LL1DiskUnforgeabilityUnchangedLemma*.

$\langle 3\rangle 4.$ CASE $LL1Restart$

We pick some variables of the appropriate types for which *LL1Restart* is true.

$\langle 4\rangle 1.$ PICK $untrustedStorage \in LL1UntrustedStorageType,$
$randomSymmetricKey \in SymmetricKeyType \setminus \{LL1NVRAM.symmetricKey\},$
$hash \in HashType :$
$LL1Restart!(untrustedStorage, randomSymmetricKey, hash)$

BY $\langle 3\rangle 4$ DEF $LL1Restart$

To prove the universally quantified expression, we take a new hash. For the TAKE step to be meaningful to the prover, first we have to tell the prover to expand the definition of *ExtendedUnforgeabilityInvariant*, so it will see the universally quantified expression therein.

$\langle 4 \rangle$ USE DEF *ExtendedUnforgeabilityInvariant*
$\langle 4 \rangle 3.$ TAKE $historyStateBinding \in HashType$
$\langle 4 \rangle 4.$ $ValidateMAC(LL1NVRAM.symmetricKey', historyStateBinding, LL1RAM.authenticator') \Rightarrow$
$LL1RAM.authenticator' \in LL1ObservedAuthenticators'$

For the RAM portion, we first note that the definition of *LL1Restart* tells us that the primed authenticator has been constructed with a random symmetric key. The antecedent of the implication cannot be true, because the random symmetric key does not match the symmetric key in the *NVRAM*. We show this using proof by contradiction.

$\langle 5 \rangle 1.$ $LL1RAM.authenticator' = GenerateMAC(randomSymmetricKey, hash)$
BY $\langle 4 \rangle 1$
$\langle 5 \rangle 2.$ $\neg ValidateMAC(LL1NVRAM.symmetricKey', historyStateBinding, LL1RAM.authenticator')$
$\langle 6 \rangle 1.$ SUFFICES
ASSUME
$ValidateMAC($
$LL1NVRAM.symmetricKey',$
$historyStateBinding,$
$LL1RAM.authenticator')$
PROVE FALSE
BY $\langle 5 \rangle 1$
$\langle 6 \rangle 2.$ $randomSymmetricKey = LL1NVRAM.symmetricKey'$
$\langle 7 \rangle 1.$ $ValidateMAC($
$LL1NVRAM.symmetricKey',$
$historyStateBinding,$
$GenerateMAC(randomSymmetricKey, hash))$
BY $\langle 5 \rangle 1$, $\langle 6 \rangle 1$
$\langle 7 \rangle 2.$ $randomSymmetricKey \in SymmetricKeyType$
BY $\langle 4 \rangle 1$
$\langle 7 \rangle 3.$ $LL1NVRAM.symmetricKey' \in SymmetricKeyType$
$\langle 8 \rangle 1.$ $LL1TypeInvariant'$
BY $\langle 2 \rangle 1$
$\langle 8 \rangle 2.$ QED
BY $\langle 8 \rangle 1$, *LL1SubtypeImplicationLemma*DEF *LL1SubtypeImplication*
$\langle 7 \rangle 4.$ $hash \in HashType$
BY $\langle 4 \rangle 1$
$\langle 7 \rangle 5.$ $historyStateBinding \in HashType$
BY $\langle 4 \rangle 3$
$\langle 7 \rangle 6.$ QED
BY $\langle 7 \rangle 1$, $\langle 7 \rangle 2$, $\langle 7 \rangle 3$, $\langle 7 \rangle 4$, $\langle 7 \rangle 5$, *MACUnforgeable*
$\langle 6 \rangle 3.$ $randomSymmetricKey \neq LL1NVRAM.symmetricKey'$
$\langle 7 \rangle 1.$ $randomSymmetricKey \neq LL1NVRAM.symmetricKey$
BY $\langle 4 \rangle 1$
$\langle 7 \rangle 2.$ UNCHANGED $LL1NVRAM.symmetricKey$
BY $\langle 2 \rangle 3$, *SymmetricKeyConstantLemma*
$\langle 7 \rangle 3.$ QED
BY $\langle 7 \rangle 1$, $\langle 7 \rangle 2$
$\langle 6 \rangle 4.$ QED
BY $\langle 6 \rangle 2$, $\langle 6 \rangle 3$
$\langle 5 \rangle 3.$ QED
BY $\langle 5 \rangle 2$

For the disk portion of the unforgeability invariant, we employ the *LL1DiskUnforgeabilityUnchangedLemma*, since the disk is not changed by the *LL1CorruptRAM* action.

$\langle 4 \rangle 5.$ $ValidateMAC(LL1NVRAM.symmetricKey', historyStateBinding, LL1Disk.authenticator') \Rightarrow$

$LL1Disk.authenticator' \in LL1ObservedAuthenticators'$

⟨5⟩1. $ExtendedUnforgeabilityInvariant \land LL1TypeInvariant \land LL1TypeInvariant'$
BY ⟨2⟩1

⟨5⟩2. UNCHANGED $LL1Disk$
BY ⟨4⟩1

⟨5⟩3. UNCHANGED $LL1ObservedAuthenticators$
BY ⟨4⟩1

⟨5⟩4. UNCHANGED $LL1NVRAM.symmetricKey$

⟨6⟩1. UNCHANGED $LL1NVRAM$
BY ⟨4⟩1

⟨6⟩2. QED
BY ⟨6⟩1

⟨5⟩5. QED
BY ⟨5⟩1, ⟨5⟩2, ⟨5⟩3, ⟨5⟩4, $LL1DiskUnforgeabilityUnchangedLemma$

⟨4⟩6. QED
BY ⟨4⟩4, ⟨4⟩5

The $LL1ReadDisk$ case is a straightforward application of the $LL1RAMUnforgeabilityUnchangedLemma$ and the $LL1DiskUnforgeabilityUnchangedLemma$.

⟨3⟩5. CASE $LL1ReadDisk$

⟨4⟩1. $\forall\, historyStateBinding \in HashType :$
$ValidateMAC(LL1NVRAM.symmetricKey', historyStateBinding, LL1RAM.authenticator') \Rightarrow$
$LL1RAM.authenticator' \in LL1ObservedAuthenticators'$

⟨5⟩1. $ExtendedUnforgeabilityInvariant \land LL1TypeInvariant \land LL1TypeInvariant'$
BY ⟨2⟩1

⟨5⟩2. $LL1RAM' = LL1Disk$
BY ⟨3⟩5 DEF $LL1ReadDisk$

⟨5⟩3. UNCHANGED $LL1ObservedAuthenticators$
BY ⟨3⟩5 DEF $LL1ReadDisk$

⟨5⟩4. UNCHANGED $LL1NVRAM.symmetricKey$

⟨6⟩1. UNCHANGED $LL1NVRAM$
BY ⟨3⟩5 DEF $LL1ReadDisk$

⟨6⟩2. QED
BY ⟨6⟩1

⟨5⟩5. QED
BY ⟨5⟩1, ⟨5⟩2, ⟨5⟩3, ⟨5⟩4, $LL1RAMUnforgeabilityUnchangedLemma$

⟨4⟩2. $\forall\, historyStateBinding \in HashType :$
$ValidateMAC(LL1NVRAM.symmetricKey', historyStateBinding, LL1Disk.authenticator') \Rightarrow$
$LL1Disk.authenticator' \in LL1ObservedAuthenticators'$

⟨5⟩1. $ExtendedUnforgeabilityInvariant \land LL1TypeInvariant \land LL1TypeInvariant'$
BY ⟨2⟩1

⟨5⟩2. UNCHANGED $LL1Disk$
BY ⟨3⟩5 DEF $LL1ReadDisk$

⟨5⟩3. UNCHANGED $LL1ObservedAuthenticators$
BY ⟨3⟩5 DEF $LL1ReadDisk$

⟨5⟩4. UNCHANGED $LL1NVRAM.symmetricKey$

⟨6⟩1. UNCHANGED $LL1NVRAM$
BY ⟨3⟩5 DEF $LL1ReadDisk$

⟨6⟩2. QED
BY ⟨6⟩1

⟨5⟩5. QED
BY ⟨5⟩1, ⟨5⟩2, ⟨5⟩3, ⟨5⟩4, $LL1DiskUnforgeabilityUnchangedLemma$

⟨4⟩3. QED

BY ⟨4⟩1, ⟨4⟩2 DEF *ExtendedUnforgeabilityInvariant*

The *LL1WriteDisk* case is a straightforward application of the *LL1RAMUnforgeabilityUnchangedLemma* and the *LL1DiskUnforgeabilityUnchangedLemma*.

⟨3⟩6. CASE *LL1WriteDisk*

⟨4⟩1. $\forall$ *historyStateBinding* $\in$ *HashType* :
ValidateMAC(*LL1NVRAM.symmetricKey′*, *historyStateBinding*, *LL1RAM.authenticator′*) $\Rightarrow$
LL1RAM.authenticator′ $\in$ *LL1ObservedAuthenticators′*

⟨5⟩1. *ExtendedUnforgeabilityInvariant* $\land$ *LL1TypeInvariant* $\land$ *LL1TypeInvariant′*
BY ⟨2⟩1

⟨5⟩2. UNCHANGED *LL1RAM*
BY ⟨3⟩6 DEF *LL1WriteDisk*

⟨5⟩3. UNCHANGED *LL1ObservedAuthenticators*
BY ⟨3⟩6 DEF *LL1WriteDisk*

⟨5⟩4. UNCHANGED *LL1NVRAM.symmetricKey*

⟨6⟩1. UNCHANGED *LL1NVRAM*
BY ⟨3⟩6 DEF *LL1WriteDisk*

⟨6⟩2. QED
BY ⟨6⟩1

⟨5⟩5. QED
BY ⟨5⟩1, ⟨5⟩2, ⟨5⟩3, ⟨5⟩4, *LL1RAMUnforgeabilityUnchangedLemma*

⟨4⟩2. $\forall$ *historyStateBinding* $\in$ *HashType* :
ValidateMAC(*LL1NVRAM.symmetricKey′*, *historyStateBinding*, *LL1Disk.authenticator′*) $\Rightarrow$
LL1Disk.authenticator′ $\in$ *LL1ObservedAuthenticators′*

⟨5⟩1. *ExtendedUnforgeabilityInvariant* $\land$ *LL1TypeInvariant* $\land$ *LL1TypeInvariant′*
BY ⟨2⟩1

⟨5⟩2. *LL1Disk′* = *LL1RAM*
BY ⟨3⟩6 DEF *LL1WriteDisk*

⟨5⟩3. UNCHANGED *LL1ObservedAuthenticators*
BY ⟨3⟩6 DEF *LL1WriteDisk*

⟨5⟩4. UNCHANGED *LL1NVRAM.symmetricKey*

⟨6⟩1. UNCHANGED *LL1NVRAM*
BY ⟨3⟩6 DEF *LL1WriteDisk*

⟨6⟩2. QED
BY ⟨6⟩1

⟨5⟩5. QED
BY ⟨5⟩1, ⟨5⟩2, ⟨5⟩3, ⟨5⟩4, *LL1DiskUnforgeabilityUnchangedLemma*

⟨4⟩3. QED
BY ⟨4⟩1, ⟨4⟩2 DEF *ExtendedUnforgeabilityInvariant*

The *LL1CorruptRAM* case is moderately interesting. The RAM portion of the unforgeability invariant is where we pull in the constraint on the set of authenticators the user can create. This constraint is explicitly expressed in the definition of the *LL1CorruptRAM* action. The disk portion is a straightforward application of the *LL1DiskUnforgeabilityUnchangedLemma*.

⟨3⟩7. CASE *LL1CorruptRAM*

We pick some variables of the appropriate types for which *LL1CorruptRAM* is true.

⟨4⟩1. PICK *untrustedStorage* $\in$ *LL1UntrustedStorageType*,
fakeSymmetricKey $\in$ *SymmetricKeyType* $\setminus$ {*LL1NVRAM.symmetricKey*},
hash $\in$ *HashType* :
LL1CorruptRAM!(*untrustedStorage*, *fakeSymmetricKey*, *hash*)
BY ⟨3⟩7 DEF *LL1CorruptRAM*

To prove the universally quantified expression, we take a new hash. For the TAKE step to be meaningful to the prover, first we have to tell the prover to expand the definition of *ExtendedUnforgeabilityInvariant*, so it will see the universally quantified expression therein.

$\langle 4 \rangle$ USE DEF *ExtendedUnforgeabilityInvariant*
$\langle 4 \rangle 3.$ TAKE $historyStateBinding \in HashType$
$\langle 4 \rangle 4.$ $ValidateMAC(LL1NVRAM.symmetricKey', historyStateBinding, LL1RAM.authenticator') \Rightarrow$
$LL1RAM.authenticator' \in LL1ObservedAuthenticators'$

For the RAM portion, we first note that the definition of *LL1CorruptRAM* tells us that the primed authenticator in the RAM is either in the unprimed set of observed authenticators or has been constructed with a fake symmetric key. We will treat these two cases separately.

$\langle 5 \rangle 1.$ $\lor$ $LL1RAM.authenticator' \in LL1ObservedAuthenticators$
$\lor$ $LL1RAM.authenticator' = GenerateMAC(fakeSymmetricKey, hash)$
BY $\langle 4 \rangle 1$

For the case in which the primed authenticator is in the unprimed set of observed authenticators, the conclusion directly follows because the set of observed authenticators is not changed by the *LL1CorruptRAM* action.

$\langle 5 \rangle 2.$ CASE $LL1RAM.authenticator' \in LL1ObservedAuthenticators$
$\langle 6 \rangle 1.$ HAVE $ValidateMAC($
$LL1NVRAM.symmetricKey', historyStateBinding, LL1RAM.authenticator')$
$\langle 6 \rangle 2.$ UNCHANGED *LL1ObservedAuthenticators*
BY $\langle 4 \rangle 1$
$\langle 6 \rangle 3.$ QED
BY $\langle 5 \rangle 2$, $\langle 6 \rangle 2$

For the case in which the primed authenticator has been constructed with a fake symmetric key, the antecedent of the implication cannot be true, because the fake symmetric key does not match the symmetric key in the *NVRAM*. We show this using proof by contradiction.

$\langle 5 \rangle 3.$ CASE $LL1RAM.authenticator' = GenerateMAC(fakeSymmetricKey, hash)$
$\langle 6 \rangle 1.$ $\neg ValidateMAC(LL1NVRAM.symmetricKey', historyStateBinding, LL1RAM.authenticator')$
$\langle 7 \rangle 1.$ SUFFICES
ASSUME
$ValidateMAC($
$LL1NVRAM.symmetricKey',$
$historyStateBinding,$
$LL1RAM.authenticator')$
PROVE FALSE
BY $\langle 5 \rangle 3$
$\langle 7 \rangle 2.$ $fakeSymmetricKey = LL1NVRAM.symmetricKey'$
$\langle 8 \rangle 1.$ $ValidateMAC($
$LL1NVRAM.symmetricKey',$
$historyStateBinding,$
$GenerateMAC(fakeSymmetricKey, hash))$
BY $\langle 5 \rangle 3$, $\langle 7 \rangle 1$
$\langle 8 \rangle 2.$ $fakeSymmetricKey \in SymmetricKeyType$
BY $\langle 4 \rangle 1$
$\langle 8 \rangle 3.$ $LL1NVRAM.symmetricKey' \in SymmetricKeyType$
$\langle 9 \rangle 1.$ $LL1TypeInvariant'$
BY $\langle 2 \rangle 1$
$\langle 9 \rangle 2.$ QED
BY $\langle 9 \rangle 1$, *LL1SubtypeImplicationLemma*DEF *LL1SubtypeImplication*
$\langle 8 \rangle 4.$ $hash \in HashType$
BY $\langle 4 \rangle 1$
$\langle 8 \rangle 5.$ $historyStateBinding \in HashType$
BY $\langle 4 \rangle 3$
$\langle 8 \rangle 6.$ QED
BY $\langle 8 \rangle 1$, $\langle 8 \rangle 2$, $\langle 8 \rangle 3$, $\langle 8 \rangle 4$, $\langle 8 \rangle 5$, *MACUnforgeable*
$\langle 7 \rangle 3.$ $fakeSymmetricKey \neq LL1NVRAM.symmetricKey'$

⟨8⟩1. $\mathit{fakeSymmetricKey} \neq \mathit{LL1NVRAM.symmetricKey}$
BY ⟨4⟩1
⟨8⟩2. UNCHANGED $\mathit{LL1NVRAM.symmetricKey}$
BY ⟨2⟩3, $\mathit{SymmetricKeyConstantLemma}$
⟨8⟩3. QED
BY ⟨8⟩1, ⟨8⟩2
⟨7⟩4. QED
BY ⟨7⟩2, ⟨7⟩3
⟨6⟩2. QED
BY ⟨6⟩1
⟨5⟩4. QED
BY ⟨5⟩1, ⟨5⟩2, ⟨5⟩3

For the disk portion of the unforgeability invariant, we employ the *LL1DiskUnforgeabilityUnchangedLemma*, since the disk is not changed by the *LL1CorruptRAM* action.

⟨4⟩5. $\mathit{ValidateMAC}(\mathit{LL1NVRAM.symmetricKey}', \mathit{historyStateBinding}, \mathit{LL1Disk.authenticator}') \Rightarrow \mathit{LL1Disk.authenticator}' \in \mathit{LL1ObservedAuthenticators}'$
⟨5⟩1. $\mathit{ExtendedUnforgeabilityInvariant} \land \mathit{LL1TypeInvariant} \land \mathit{LL1TypeInvariant}'$
BY ⟨2⟩1
⟨5⟩2. UNCHANGED $\mathit{LL1Disk}$
BY ⟨4⟩1
⟨5⟩3. UNCHANGED $\mathit{LL1ObservedAuthenticators}$
BY ⟨4⟩1
⟨5⟩4. UNCHANGED $\mathit{LL1NVRAM.symmetricKey}$
⟨6⟩1. UNCHANGED $\mathit{LL1NVRAM}$
BY ⟨4⟩1
⟨6⟩2. QED
BY ⟨6⟩1
⟨5⟩5. QED
BY ⟨5⟩1, ⟨5⟩2, ⟨5⟩3, ⟨5⟩4, $\mathit{LL1DiskUnforgeabilityUnchangedLemma}$
⟨4⟩6. QED
BY ⟨4⟩4, ⟨4⟩5

The *LL1RestrictedCorruption* case is moderately interesting. For the RAM portion, we have two cases depending on whether the *LL1RestrictedCorruption* action leaves the RAM unchanged or trashes it in the same way an *LL1Restart* action does. For the latter case, we pull in the constraint on the set of state authenticators that can wind up in the randomized RAM after the action completes. The disk portion is a straightforward application of the *LL1DiskUnforgeabilityUnchangedLemma*.

⟨3⟩8. CASE $\mathit{LL1RestrictedCorruption}$

To prove the universally quantified expression, we take a new hash. For the TAKE step to be meaningful to the prover, first we have to tell the prover to expand the definition of *ExtendedUnforgeabilityInvariant*, so it will see the universally quantified expression therein.

⟨4⟩ USE DEF $\mathit{ExtendedUnforgeabilityInvariant}$
⟨4⟩1. TAKE $\mathit{historyStateBinding} \in \mathit{HashType}$
⟨4⟩2. $\mathit{ValidateMAC}(\mathit{LL1NVRAM.symmetricKey}', \mathit{historyStateBinding}, \mathit{LL1RAM.authenticator}') \Rightarrow \mathit{LL1RAM.authenticator}' \in \mathit{LL1ObservedAuthenticators}'$

First we consider the case in which the RAM is unchanged. This is straightforward

⟨5⟩1. CASE $\mathit{LL1RestrictedCorruption!ram!unchanged}$
⟨6⟩1. $\mathit{ExtendedUnforgeabilityInvariant} \land \mathit{LL1TypeInvariant} \land \mathit{LL1TypeInvariant}'$
BY ⟨2⟩1
⟨6⟩2. UNCHANGED $\mathit{LL1RAM}$
BY ⟨6⟩1
⟨6⟩3. UNCHANGED $\mathit{LL1ObservedAuthenticators}$
BY ⟨3⟩8 DEF $\mathit{LL1RestrictedCorruption}$

$\langle 6\rangle 4.$ UNCHANGED $LL1NVRAM.symmetricKey$
BY $\langle 2\rangle 3$, $SymmetricKeyConstantLemma$
$\langle 6\rangle 5.$ QED
BY $\langle 6\rangle 1$, $\langle 6\rangle 2$, $\langle 6\rangle 3$, $\langle 6\rangle 4$, $LL1RAMUnforgeabilityUnchangedLemma$

Next we consider the case in which the RAM is *trashed*.

$\langle 5\rangle 2.$ CASE $LL1RestrictedCorruption!ram!trashed$

We pick some variables of the appropriate types for which $LL1RestrictedCorruption$ is true.

$\langle 6\rangle 1.$ PICK $untrustedStorage \in LL1UntrustedStorageType$,
$randomSymmetricKey \in SymmetricKeyType \setminus \{LL1NVRAM.symmetricKey\}$,
$hash \in HashType$:
$LL1RestrictedCorruption!ram!trashed!($
$untrustedStorage, randomSymmetricKey, hash)$
BY $\langle 5\rangle 2$

The primed authenticator has been constructed with a random symmetric key. The antecedent of the implication cannot be true, because the random symmetric key does not match the symmetric key in the $NVRAM$. We show this using proof by contradiction.

$\langle 6\rangle 2.$ $LL1RAM.authenticator' = GenerateMAC(randomSymmetricKey, hash)$
BY $\langle 6\rangle 1$
$\langle 6\rangle 3.$ $\neg ValidateMAC(LL1NVRAM.symmetricKey', historyStateBinding, LL1RAM.authenticator')$
$\langle 7\rangle 1.$ SUFFICES
ASSUME
$ValidateMAC($
$LL1NVRAM.symmetricKey',$
$historyStateBinding,$
$LL1RAM.authenticator')$
PROVE FALSE
BY $\langle 6\rangle 1$
$\langle 7\rangle 2.$ $randomSymmetricKey = LL1NVRAM.symmetricKey'$
$\langle 8\rangle 1.$ $ValidateMAC($
$LL1NVRAM.symmetricKey',$
$historyStateBinding,$
$GenerateMAC(randomSymmetricKey, hash))$
BY $\langle 6\rangle 1$, $\langle 7\rangle 1$
$\langle 8\rangle 2.$ $randomSymmetricKey \in SymmetricKeyType$
BY $\langle 6\rangle 1$
$\langle 8\rangle 3.$ $LL1NVRAM.symmetricKey' \in SymmetricKeyType$
$\langle 9\rangle 1.$ $LL1TypeInvariant'$
BY $\langle 2\rangle 1$
$\langle 9\rangle 2.$ QED
BY $\langle 9\rangle 1$, $LL1SubtypeImplicationLemma$DEF $LL1SubtypeImplication$
$\langle 8\rangle 4.$ $hash \in HashType$
BY $\langle 6\rangle 1$
$\langle 8\rangle 5.$ $historyStateBinding \in HashType$
BY $\langle 6\rangle 3$
$\langle 8\rangle 6.$ QED
BY $\langle 8\rangle 1$, $\langle 8\rangle 2$, $\langle 8\rangle 3$, $\langle 8\rangle 4$, $\langle 8\rangle 5$, $MACUnforgeable$
$\langle 7\rangle 3.$ $randomSymmetricKey \neq LL1NVRAM.symmetricKey'$
$\langle 8\rangle 1.$ $randomSymmetricKey \neq LL1NVRAM.symmetricKey$
BY $\langle 6\rangle 1$
$\langle 8\rangle 2.$ UNCHANGED $LL1NVRAM.symmetricKey$
BY $\langle 2\rangle 3$, $SymmetricKeyConstantLemma$
$\langle 8\rangle 3.$ QED

BY $\langle 8\rangle 1$, $\langle 8\rangle 2$

$\langle 7\rangle 4$. QED

BY $\langle 7\rangle 2$, $\langle 7\rangle 3$

$\langle 6\rangle 4$. QED

BY $\langle 6\rangle 3$

$\langle 5\rangle 3$. QED

BY $\langle 3\rangle 8$, $\langle 5\rangle 1$, $\langle 5\rangle 2$ DEF *LL1RestrictedCorruption*

For the disk portion of the unforgeability invariant, we employ the *LL1DiskUnforgeabilityUnchangedLemma*, since the disk is not changed by the *LL1CorruptRAM* action.

$\langle 4\rangle 3$. $ValidateMAC(LL1NVRAM.symmetricKey', historyStateBinding, LL1Disk.authenticator') \Rightarrow$
$LL1Disk.authenticator' \in LL1ObservedAuthenticators'$

$\langle 5\rangle 1$. $ExtendedUnforgeabilityInvariant \land LL1TypeInvariant \land LL1TypeInvariant'$

BY $\langle 2\rangle 1$

$\langle 5\rangle 2$. UNCHANGED *LL1Disk*

BY $\langle 3\rangle 8$ DEF *LL1RestrictedCorruption*

$\langle 5\rangle 3$. UNCHANGED *LL1ObservedAuthenticators*

BY $\langle 3\rangle 8$ DEF *LL1RestrictedCorruption*

$\langle 5\rangle 4$. UNCHANGED *LL1NVRAM.symmetricKey*

BY $\langle 2\rangle 3$, *SymmetricKeyConstantLemma*

$\langle 5\rangle 5$. QED

BY $\langle 5\rangle 1$, $\langle 5\rangle 2$, $\langle 5\rangle 3$, $\langle 5\rangle 4$, *LL1DiskUnforgeabilityUnchangedLemma*

$\langle 4\rangle 4$. QED

BY $\langle 4\rangle 2$, $\langle 4\rangle 3$

$\langle 3\rangle 9$. QED

BY $\langle 2\rangle 3$, $\langle 3\rangle 1$, $\langle 3\rangle 2$, $\langle 3\rangle 3$, $\langle 3\rangle 4$, $\langle 3\rangle 5$, $\langle 3\rangle 6$, $\langle 3\rangle 7$, $\langle 3\rangle 8$ DEF *LL1Next*

$\langle 2\rangle 4$. QED

BY $\langle 2\rangle 1$, $\langle 2\rangle 2$, $\langle 2\rangle 3$

$\langle 1\rangle 4$. QED

Using the *Inv*1 proof rule, the base case and the induction step together imply that the invariant always holds.

$\langle 2\rangle 1$. $(\land\ ExtendedUnforgeabilityInvariant$
$\land\ \Box[LL1Next]_{LL1Vars}$
$\land\ \Box LL1TypeInvariant)$
$\Rightarrow$
$\Box ExtendedUnforgeabilityInvariant$

BY $\langle 1\rangle 3$, *Inv*1

$\langle 2\rangle 2$. QED

BY $\langle 1\rangle 1$, $\langle 1\rangle 2$, $\langle 2\rangle 1$ DEF *LL1Spec*

The *UnforgeabilityInvariant* follows directly from the *ExtendedUnforgeabilityInvariant*.

THEOREM $UnforgeabilityInvariance \triangleq LL1Spec \Rightarrow \Box UnforgeabilityInvariant$

$\langle 1\rangle 1$. $LL1Spec \Rightarrow \Box ExtendedUnforgeabilityInvariant$

BY *ExtendedUnforgeabilityInvariance*

$\langle 1\rangle 2$. $ExtendedUnforgeabilityInvariant \Rightarrow UnforgeabilityInvariant$

BY DEF *ExtendedUnforgeabilityInvariant*, *UnforgeabilityInvariant*

$\langle 1\rangle 3$. QED

BY $\langle 1\rangle 1$, $\langle 1\rangle 2$

4.6 Proof of Inclusion, Cardinality, and Uniqueness Co-invariance in Memoir-Basic

MODULE *MemoirLL1InclCardUniqInvariance*

This module proves that the *InclusionInvariant*, the *CardinalityInvariant*, and the *UniquenessInvariant* are all inductive invariants of the Memoir-Basic spec. Then, from this proof and the previous proof that the *UnforgeabilityInvariant* is an inductive invariant of the Memoir-Basic spec, it proves that the set of *CorrectnessInvariants* are inductive invariants of the Memoir-Basic spec.

EXTENDS *MemoirLL1UnforgeabilityInvariance*

These three invariants cannot be ordered with respect to each other. The proof of $InclusionInvariant'$ inductively depends upon *UniquenessInvariant*; the proof of $UniquenessInvariant'$ inductively depends upon *CardinalityInvariant*; and the proof of $CardinalityInvariant'$ inductively depends upon *InclusionInvariant*. Therefore, we have to prove them co-inductively.

THEOREM $InclusionCardinalityUniquenessInvariance \triangleq$
$LL1Spec \Rightarrow \Box InclusionInvariant \land \Box CardinalityInvariant \land \Box UniquenessInvariant$

This proof will require the *LL1TypeInvariant* and the *UnforgeabilityInvariant*. Fortunately, the *LL1TypeSafe* theorem has already proven that the Memoir-Basic spec satisfies its type invariant, and the *UnforgeabilityInvariance* theorem has already proven that the Memoir-Basic spec satisfies the *UnforgeabilityInvariant*.

$\langle 1 \rangle 1.$ $LL1Spec \Rightarrow \Box LL1TypeInvariant$
BY *LL1TypeSafe*
$\langle 1 \rangle 2.$ $LL1Spec \Rightarrow \Box UnforgeabilityInvariant$
BY *UnforgeabilityInvariance*

The top level of the proof is boilerplate TLA+ for an *Inv*1-style proof. First, we prove that the initial state satisfies the three invariants. Second, we prove that the *LL1Next* predicate inductively preserves the three invariants. Third, we use temporal induction to prove that these two conditions satisfy the three invariants over all behaviors.

$\langle 1 \rangle 3.$ $LL1Init \land LL1TypeInvariant \land UnforgeabilityInvariant \Rightarrow$
$InclusionInvariant \land CardinalityInvariant \land UniquenessInvariant$

First, we assume the antecedents.

$\langle 2 \rangle 1.$ HAVE $LL1Init \land LL1TypeInvariant$

Then, we pick some symmetrick key for which *LL1Init* is true.

$\langle 2 \rangle 2.$ PICK $symmetricKey \in SymmetricKeyType : LL1Init!(symmetricKey)!1$
BY $\langle 2 \rangle 1$ DEF *LL1Init*

To simplify the writing of the proof, we re-state the definitions from *LL1Init*.

$\langle 2 \rangle$ $initialPrivateStateEnc \triangleq SymmetricEncrypt(symmetricKey, InitialPrivateState)$
$\langle 2 \rangle$ $initialStateHash \triangleq Hash(InitialPublicState, initialPrivateStateEnc)$
$\langle 2 \rangle$ $initialHistoryStateBinding \triangleq Hash(BaseHashValue, initialStateHash)$
$\langle 2 \rangle$ $initialAuthenticator \triangleq GenerateMAC(symmetricKey, initialHistoryStateBinding)$
$\langle 2 \rangle$ $initialUntrustedStorage \triangleq [$
$publicState \mapsto InitialPublicState,$
$privateStateEnc \mapsto initialPrivateStateEnc,$
$historySummary \mapsto BaseHashValue,$
$authenticator \mapsto initialAuthenticator]$
$\langle 2 \rangle$ $initialTrustedStorage \triangleq [$
$historySummary \mapsto BaseHashValue,$
$symmetricKey \mapsto symmetricKey]$

We then assert the type safety of these definitions, with the help of the *LL1InitDefsTypeSafeLemma*.

$\langle 2 \rangle 3.$ $\land$ $initialPrivateStateEnc \in PrivateStateEncType$
$\land$ $initialStateHash \in HashType$
$\land$ $initialHistoryStateBinding \in HashType$
$\land$ $initialAuthenticator \in MACType$

$\wedge$ *initialUntrustedStorage* $\in$ *LL1UntrustedStorageType*
$\wedge$ *initialTrustedStorage* $\in$ *LL1TrustedStorageType*

⟨3⟩1. *symmetricKey* $\in$ *SymmetricKeyType*
BY ⟨2⟩2
⟨3⟩2. QED
BY ⟨3⟩1, *LL1InitDefsTypeSafeLemma*

We hide the definitions, so they don't overwhelm the prover. We'll pull them in as necessary below.

⟨2⟩ HIDE DEF *initialPrivateStateEnc*, *initialStateHash*, *initialHistoryStateBinding*, *initialAuthenticator*, *initialUntrustedStorage*, *initialTrustedStorage*

We'll prove each of the three invariants separately. First, we'll prove the *InclusionInvariant*.

⟨2⟩4. *InclusionInvariant*

To prove the universally quantified expression, we take a new set of variables of the appropriate types. For the TAKE step to be meaningful to the prover, first we have to tell the prover to expand the definition of *InclusionInvariant*, so it will see the universally quantified expression therein.

⟨3⟩ USE DEF *InclusionInvariant*
⟨3⟩1. TAKE *input* $\in$ *InputType*,
historySummary $\in$ *HashType*,
publicState $\in$ *PublicStateType*,
privateStateEnc $\in$ *PrivateStateEncType*

It suffices to prove that the the antecedent of the implication is false, which merely requires showing that one of the conjuncts is false. The proof is straightforward, since the history summary in the *NVRAM* equals the base hash value, and the base hash value cannot be constructed as the hash of any other values.

⟨3⟩2. SUFFICES ASSUME TRUEPROVE *LL1NVRAM.historySummary* $\neq$ *Hash*(*historySummary*, *input*)
OBVIOUS
⟨3⟩3. *LL1NVRAM.historySummary* = *BaseHashValue*
⟨4⟩1. *LL1NVRAM* = [*historySummary* $\mapsto$ *BaseHashValue*,
symmetricKey $\mapsto$ *symmetricKey*]
BY ⟨2⟩2
⟨4⟩2. QED
BY ⟨4⟩1
⟨3⟩4. *historySummary* $\in$ *HashDomain*
⟨4⟩1. *historySummary* $\in$ *HashType*
BY ⟨3⟩1
⟨4⟩2. QED
BY ⟨4⟩1 DEF *HashDomain*
⟨3⟩5. *input* $\in$ *HashDomain*
⟨4⟩1. *input* $\in$ *InputType*
BY ⟨3⟩1
⟨4⟩2. QED
BY ⟨4⟩1 DEF *HashDomain*
⟨3⟩6. QED
BY ⟨3⟩3, ⟨3⟩4, ⟨3⟩5, *BaseHashValueUnique*

Second, we'll prove the *CardinalityInvariant*.

⟨2⟩5. *CardinalityInvariant*

To prove the universally quantified expression, we take a new set of variables of the appropriate types. For the TAKE step to be meaningful to the prover, first we have to tell the prover to expand the definition of *CardinalityInvariant*, so it will see the universally quantified expression therein.

⟨3⟩ USE DEF *CardinalityInvariant*
⟨3⟩1. TAKE *historySummary* $\in$ *HashType*, *stateHash* $\in$ *HashType*

To simplify the writing of the proof, we re-state the definition from the *CardinalityInvariant*.

⟨3⟩ *historyStateBinding* $\triangleq$ *Hash*(*historySummary*, *stateHash*)

We then assert the type safety of this definition, with the help of the *CardinalityInvariantDefsTypeSafeLemma*.

$\langle 3 \rangle 2.$ *historyStateBinding* $\in$ *HashType*
BY $\langle 3 \rangle 1$, *CardinalityInvariantDefsTypeSafeLemma*

To prove the implication, it suffices to assume the antecedent and prove the consequent.

$\langle 3 \rangle 3.$ SUFFICES
ASSUME
$\land$ *LL1NVRAMHistorySummaryUncorrupted*
$\land$ *LL1HistoryStateBindingAuthenticated*(*historyStateBinding*)
PROVE
HashCardinality(*historySummary*) $\leq$ *HashCardinality*(*LL1NVRAM.historySummary*)
OBVIOUS

We hide the definition, so it doesn't overwhelm the prover. We'll pull it in as necessary below.

$\langle 3 \rangle$ HIDE DEF *historyStateBinding*

The proof is simple. First, we prove that the hash cardinality of the history summary is zero.

$\langle 3 \rangle 4.$ *HashCardinality*(*historySummary*) $= 0$

There are two steps to proving that the hash cardinality of the history summary is zero. First, we use the *MACCollisionResistant* property to prove that the history state binding matches the initial history state binding defined in *LL1Init*.

$\langle 4 \rangle 2.$ *historyStateBinding* $=$ *initialHistoryStateBinding*
$\langle 5 \rangle 1.$ *ValidateMAC*(*LL1NVRAM.symmetricKey*, *historyStateBinding*, *initialAuthenticator*)
$\langle 6 \rangle 1.$ *LL1ObservedAuthenticators* $= \{$*initialAuthenticator*$\}$
BY $\langle 2 \rangle 2$ DEF *initialAuthenticator*, *initialHistoryStateBinding*, *initialStateHash*, *initialPrivateStateEnc*
$\langle 6 \rangle 2.$ *LL1HistoryStateBindingAuthenticated*(*historyStateBinding*)
BY $\langle 3 \rangle 3$
$\langle 6 \rangle 3.$ QED
BY $\langle 6 \rangle 1$, $\langle 6 \rangle 2$ DEF *LL1HistoryStateBindingAuthenticated*
$\langle 5 \rangle 2.$ *LL1NVRAM.symmetricKey* $\in$ *SymmetricKeyType*
$\langle 6 \rangle 1.$ *LL1TypeInvariant*
BY $\langle 2 \rangle 1$
$\langle 6 \rangle 2.$ QED
BY $\langle 6 \rangle 1$, *LL1SubtypeImplicationLemma*DEF *LL1SubtypeImplication*
$\langle 5 \rangle 3.$ *symmetricKey* $\in$ *SymmetricKeyType*
BY $\langle 2 \rangle 2$
$\langle 5 \rangle 4.$ *historyStateBinding* $\in$ *HashType*
BY $\langle 3 \rangle 2$
$\langle 5 \rangle 5.$ *initialHistoryStateBinding* $\in$ *HashType*
BY $\langle 2 \rangle 3$
$\langle 5 \rangle 6.$ QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$, $\langle 5 \rangle 3$, $\langle 5 \rangle 4$, $\langle 5 \rangle 5$, *MACCollisionResistant*
DEF *initialAuthenticator*

Second, we use the *HashCollisionResistant* property to prove that the history summary matches the *BaseHashValue*, which is the initial history summary in *LL1Init*.

$\langle 4 \rangle 3.$ *historySummary* $=$ *BaseHashValue*
$\langle 5 \rangle 1.$ *historySummary* $\in$ *HashDomain*
$\langle 6 \rangle 1.$ *historySummary* $\in$ *HashType*
BY $\langle 3 \rangle 1$
$\langle 6 \rangle 2.$ QED
BY $\langle 6 \rangle 1$ DEF *HashDomain*
$\langle 5 \rangle 2.$ *stateHash* $\in$ *HashDomain*
$\langle 6 \rangle 1.$ *stateHash* $\in$ *HashType*

BY ⟨3⟩1
⟨6⟩2. QED
BY ⟨6⟩1 DEF *HashDomain*
⟨5⟩3. $BaseHashValue \in HashDomain$
BY *BaseHashValueTypeSafe*DEF *HashDomain*
⟨5⟩4. $initialStateHash \in HashDomain$
⟨6⟩1. $initialStateHash \in HashType$
BY ⟨2⟩3
⟨6⟩2. QED
BY ⟨6⟩1 DEF *HashDomain*
⟨5⟩5. QED
BY ⟨4⟩2, ⟨5⟩1, ⟨5⟩2, ⟨5⟩3, ⟨5⟩4, *HashCollisionResistant*
DEF *historyStateBinding*, *initialHistoryStateBinding*

The conclusion follows directly, because the *BaseHashValue* has cardinality zero.

⟨4⟩4. QED
BY ⟨4⟩3, *BaseHashCardinalityZero*

Then, we prove that the hash cardinality of the history summary in the *NVRAM* is zero.

⟨3⟩5. $HashCardinality(LL1NVRAM.historySummary) = 0$
⟨4⟩1. $LL1NVRAM.historySummary = BaseHashValue$
⟨5⟩1. $LL1NVRAM = [historySummary \mapsto BaseHashValue, symmetricKey \mapsto symmetricKey]$
BY ⟨2⟩2
⟨5⟩2. QED
BY ⟨5⟩1
⟨4⟩2. QED
BY ⟨4⟩1, *BaseHashCardinalityZero*

Since zero is less than or equal to zero, the *CardinalityInvariant* holds.

⟨3⟩6. QED
BY ⟨3⟩4, ⟨3⟩5

Third, we'll prove the *UniquenessInvariant*.

⟨2⟩6. *UniquenessInvariant*

To prove the universally quantified expression, we take a new set of variables of the appropriate types. For the TAKE step to be meaningful to the prover, first we have to tell the prover to expand the definition of *UniquenessInvariant*, so it will see the universally quantified expression therein.

⟨3⟩ USE DEF *UniquenessInvariant*
⟨3⟩1. TAKE $stateHash1, stateHash2 \in HashType$

To simplify the writing of the proof, we re-state the definitions from the *UniquenessInvariant*.

⟨3⟩ $historyStateBinding1 \triangleq Hash(LL1NVRAM.historySummary, stateHash1)$
⟨3⟩ $historyStateBinding2 \triangleq Hash(LL1NVRAM.historySummary, stateHash2)$

We then assert the type safety of these definitions, with the help of the *UniquenessInvariantDefsTypeSafeLemma*.

⟨3⟩2. $\land\ historyStateBinding1 \in HashType$
$\land\ historyStateBinding2 \in HashType$
⟨4⟩1. *LL1TypeInvariant*
BY ⟨2⟩1
⟨4⟩2. QED
BY ⟨3⟩1, ⟨4⟩1, *UniquenessInvariantDefsTypeSafeLemma*

To prove the implication, it suffices to assume the antecedent and prove the consequent.

⟨3⟩3. SUFFICES
ASSUME $\land\ LL1HistoryStateBindingAuthenticated(historyStateBinding1)$
$\land\ LL1HistoryStateBindingAuthenticated(historyStateBinding2)$

PROVE $stateHash1 = stateHash2$
OBVIOUS

We hide the definitions, so they don't overwhelm the prover. We'll pull them in as necessary below.

$\langle 3 \rangle$ HIDE DEF $historyStateBinding1$, $historyStateBinding2$

Then, we'll prove that the two history state bindings are equal to each other, because each one is equal to the initial state binding.

$\langle 3 \rangle 4.$ $historyStateBinding1 = historyStateBinding2$

To prove that history state binding 1 equals the initial history state binding defined in *LL1Init*, we'll appeal to the *MACCollisionResistant* property.

$\langle 4 \rangle 1.$ $historyStateBinding1 = initialHistoryStateBinding$

The main precondition for *MACCollisionResistant* is that *historyStateBinding*1 is validated by an authenticator that was generated with *initialHistoryStateBinding*.

$\langle 5 \rangle 1.$ $ValidateMAC(LL1NVRAM.symmetricKey, historyStateBinding1, initialAuthenticator)$
$\langle 6 \rangle 1.$ $LL1ObservedAuthenticators = \{initialAuthenticator\}$
BY $\langle 2 \rangle 2$ DEF $initialAuthenticator$, $initialHistoryStateBinding$, $initialStateHash$, $initialPrivateStateEnc$
$\langle 6 \rangle 2.$ $LL1HistoryStateBindingAuthenticated(historyStateBinding1)$
BY $\langle 3 \rangle 3$
$\langle 6 \rangle 3.$ QED
BY $\langle 6 \rangle 1$, $\langle 6 \rangle 2$ DEF $LL1HistoryStateBindingAuthenticated$

We also have to prove the appropriate types, which are the next four statements.

$\langle 5 \rangle 2.$ $LL1NVRAM.symmetricKey \in SymmetricKeyType$
$\langle 6 \rangle 1.$ $LL1TypeInvariant$
BY $\langle 2 \rangle 1$
$\langle 6 \rangle 2.$ QED
BY $\langle 6 \rangle 1$, $LL1SubtypeImplicationLemma$DEF $LL1SubtypeImplication$
$\langle 5 \rangle 3.$ $symmetricKey \in SymmetricKeyType$
BY $\langle 2 \rangle 2$
$\langle 5 \rangle 4.$ $historyStateBinding1 \in HashType$
BY $\langle 3 \rangle 2$
$\langle 5 \rangle 5.$ $initialHistoryStateBinding \in HashType$
BY $\langle 2 \rangle 3$

Now, we can apply the *MACCollisionResistant* property, which establishes that history state binding 1 equals the initial history state binding.

$\langle 5 \rangle 6.$ QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$, $\langle 5 \rangle 3$, $\langle 5 \rangle 4$, $\langle 5 \rangle 5$, $MACCollisionResistant$
DEF $initialAuthenticator$

To prove that history state binding 2 equals the initial history state binding defined in *LL1Init*, we'll appeal to the *MACCollisionResistant* property. This is the exact same logic we followed above for history state binding 1.

$\langle 4 \rangle 2.$ $historyStateBinding2 = initialHistoryStateBinding$

The main precondition for *MACCollisionResistant* is that *historyStateBinding*1 is validated by an authenticator that was generated with *initialHistoryStateBinding*.

$\langle 5 \rangle 1.$ $ValidateMAC(LL1NVRAM.symmetricKey, historyStateBinding2, initialAuthenticator)$
$\langle 6 \rangle 1.$ $LL1ObservedAuthenticators = \{initialAuthenticator\}$
BY $\langle 2 \rangle 2$ DEF $initialAuthenticator$, $initialHistoryStateBinding$, $initialStateHash$, $initialPrivateStateEnc$
$\langle 6 \rangle 2.$ $LL1HistoryStateBindingAuthenticated(historyStateBinding2)$
BY $\langle 3 \rangle 3$
$\langle 6 \rangle 3.$ QED
BY $\langle 6 \rangle 1$, $\langle 6 \rangle 2$ DEF $LL1HistoryStateBindingAuthenticated$

We also have to prove the appropriate types, which are the next four statements.

$\langle 5\rangle 2.$ $LL1NVRAM.symmetricKey \in SymmetricKeyType$
$\langle 6\rangle 1.$ $LL1TypeInvariant$
BY $\langle 2\rangle 1$
$\langle 6\rangle 2.$ QED
BY $\langle 6\rangle 1$, $LL1SubtypeImplicationLemma$DEF $LL1SubtypeImplication$
$\langle 5\rangle 3.$ $symmetricKey \in SymmetricKeyType$
BY $\langle 2\rangle 2$
$\langle 5\rangle 4.$ $historyStateBinding2 \in HashType$
BY $\langle 3\rangle 2$
$\langle 5\rangle 5.$ $initialHistoryStateBinding \in HashType$
BY $\langle 2\rangle 3$

Now, we can apply the *MACCollisionResistant* property, which establishes that history state binding 2 equals the initial history state binding.

$\langle 5\rangle 6.$ QED
BY $\langle 5\rangle 1$, $\langle 5\rangle 2$, $\langle 5\rangle 3$, $\langle 5\rangle 4$, $\langle 5\rangle 5$, $MACCollisionResistant$
DEF $initialAuthenticator$

Because each history state binding is equal to the initial state binding, the two history state bindings must equal each other.

$\langle 4\rangle 3.$ QED
BY $\langle 4\rangle 1$, $\langle 4\rangle 2$

Since the two state bindings are equal, it follows from the collision resistance of the hash function that the two state hashes are equal.

$\langle 3\rangle 5.$ QED
$\langle 4\rangle 1.$ $stateHash1 \in HashDomain$
$\langle 5\rangle 1.$ $stateHash1 \in HashType$
BY $\langle 3\rangle 1$
$\langle 5\rangle 2.$ QED
BY $\langle 2\rangle 1$ DEF $HashDomain$
$\langle 4\rangle 2.$ $stateHash2 \in HashDomain$
$\langle 5\rangle 1.$ $stateHash2 \in HashType$
BY $\langle 3\rangle 1$
$\langle 5\rangle 2.$ QED
BY $\langle 2\rangle 1$ DEF $HashDomain$
$\langle 4\rangle 3.$ $LL1NVRAM.historySummary \in HashDomain$
$\langle 5\rangle 1.$ $LL1NVRAM.historySummary \in HashType$
$\langle 6\rangle 1.$ $LL1TypeInvariant$
BY $\langle 2\rangle 1$
$\langle 6\rangle 2.$ QED
BY $\langle 6\rangle 1$, $LL1SubtypeImplicationLemma$DEF $LL1SubtypeImplication$
$\langle 5\rangle 2.$ QED
BY $\langle 5\rangle 1$ DEF $HashDomain$
$\langle 4\rangle 4.$ QED
BY $\langle 3\rangle 4$, $\langle 4\rangle 1$, $\langle 4\rangle 2$, $\langle 4\rangle 3$, $HashCollisionResistant$
DEF $historyStateBinding1$, $historyStateBinding2$
$\langle 2\rangle 7.$ QED
BY $\langle 2\rangle 4$, $\langle 2\rangle 5$, $\langle 2\rangle 6$ DEF $CorrectnessInvariants$

For the induction step, we will need the type invariant and the unforgeablity invariant to be true in both the unprimed and primed states.

$\langle 1\rangle 4.$ ($\land\ InclusionInvariant$
$\land\ CardinalityInvariant$
$\land\ UniquenessInvariant$
$\land\ [LL1Next]_{LL1Vars}$

$\wedge$ *LL1TypeInvariant*
$\wedge$ *LL1TypeInvariant'*
$\wedge$ *UnforgeabilityInvariant*
$\wedge$ *UnforgeabilityInvariant'*)
$\Rightarrow$
InclusionInvariant' $\wedge$ *CardinalityInvariant'* $\wedge$ *UniquenessInvariant'*

First, we assume the antecedents.

⟨2⟩1. HAVE $\wedge$ *InclusionInvariant*
$\wedge$ *CardinalityInvariant*
$\wedge$ *UniquenessInvariant*
$\wedge$ $[LL1Next]_{LL1Vars}$
$\wedge$ *LL1TypeInvariant*
$\wedge$ *LL1TypeInvariant'*
$\wedge$ *UnforgeabilityInvariant*
$\wedge$ *UnforgeabilityInvariant'*

The induction step includes two cases: stuttering and *LL1Next* actions. The stuttering case is a straightforward application of the *InclusionUnchangedLemma*, the *CardinalityUnchangedLemma*, and the *UniquenessUnchangedLemma*.

⟨2⟩2. CASE UNCHANGED *LL1Vars*
⟨3⟩1. UNCHANGED ⟨*LL1NVRAM*, *LL1ObservedOutputs*⟩
BY ⟨2⟩2 DEF *LL1Vars*
⟨3⟩2. $\forall$ *historyStateBinding* $\in$ *HashType* :
UNCHANGED *LL1HistoryStateBindingAuthenticated*(*historyStateBinding*)
⟨4⟩1. UNCHANGED ⟨*LL1NVRAM*, *LL1ObservedAuthenticators*⟩
BY ⟨2⟩2 DEF *LL1Vars*
⟨4⟩2. QED
BY ⟨4⟩1, *UnchangedAuthenticatedHistoryStateBindingsLemma*
⟨3⟩3. *InclusionInvariant'*
⟨4⟩1. *InclusionInvariant* $\wedge$ *LL1TypeInvariant* $\wedge$ *LL1TypeInvariant'*
BY ⟨2⟩1
⟨4⟩2. QED
BY ⟨3⟩1, ⟨3⟩2, ⟨4⟩1, *InclusionUnchangedLemma*
⟨3⟩4. *CardinalityInvariant'*
⟨4⟩1. *CardinalityInvariant* $\wedge$ *LL1TypeInvariant*
BY ⟨2⟩1
⟨4⟩2. QED
BY ⟨3⟩1, ⟨3⟩2, ⟨4⟩1, *CardinalityUnchangedLemma*
⟨3⟩5. *UniquenessInvariant'*
⟨4⟩1. *UniquenessInvariant* $\wedge$ *LL1TypeInvariant*
BY ⟨2⟩1
⟨4⟩2. QED
BY ⟨3⟩1, ⟨3⟩2, ⟨4⟩1, *UniquenessUnchangedLemma*
⟨3⟩6. QED
BY ⟨3⟩3, ⟨3⟩4, ⟨3⟩5

We break down the *LL1Next* case into separate cases for each action.

⟨2⟩3. CASE *LL1Next*

The *LL1MakeInputAvailable* case is a straightforward application of the *InclusionUnchangedLemma*, the *CardinalityUnchangedLemma*, and the *UniquenessUnchangedLemma*.

⟨3⟩1. CASE *LL1MakeInputAvailable*
⟨4⟩1. UNCHANGED ⟨*LL1NVRAM*, *LL1ObservedOutputs*⟩
BY ⟨3⟩1 DEF *LL1MakeInputAvailable*

$\langle 4 \rangle 2.$ $\forall$ *historyStateBinding* $\in$ *HashType* :
UNCHANGED *LL1HistoryStateBindingAuthenticated*(*historyStateBinding*)
$\langle 5 \rangle 1.$ UNCHANGED $\langle$*LL1NVRAM*, *LL1ObservedAuthenticators*$\rangle$
BY $\langle 3 \rangle 1$ DEF *LL1MakeInputAvailable*
$\langle 5 \rangle 2.$ QED
BY $\langle 5 \rangle 1$, *UnchangedAuthenticatedHistoryStateBindingsLemma*
$\langle 4 \rangle 3.$ *InclusionInvariant'*
$\langle 5 \rangle 1.$ *InclusionInvariant* $\wedge$ *LL1TypeInvariant* $\wedge$ *LL1TypeInvariant'*
BY $\langle 2 \rangle 1$
$\langle 5 \rangle 2.$ QED
BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$, $\langle 5 \rangle 1$, *InclusionUnchangedLemma*
$\langle 4 \rangle 4.$ *CardinalityInvariant'*
$\langle 5 \rangle 1.$ *CardinalityInvariant* $\wedge$ *LL1TypeInvariant*
BY $\langle 2 \rangle 1$
$\langle 5 \rangle 2.$ QED
BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$, $\langle 5 \rangle 1$, *CardinalityUnchangedLemma*
$\langle 4 \rangle 5.$ *UniquenessInvariant'*
$\langle 5 \rangle 1.$ *UniquenessInvariant* $\wedge$ *LL1TypeInvariant*
BY $\langle 2 \rangle 1$
$\langle 5 \rangle 2.$ QED
BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$, $\langle 5 \rangle 1$, *UniquenessUnchangedLemma*
$\langle 4 \rangle 6.$ QED
BY $\langle 4 \rangle 3$, $\langle 4 \rangle 4$, $\langle 4 \rangle 5$

The *LL1PerformOperation* case is the vast majority of this proof.

$\langle 3 \rangle 2.$ CASE *LL1PerformOperation*

We pick some input for which *LL1PerformOperation* is true.

$\langle 4 \rangle 1.$ PICK *input1* $\in$ *LL1AvailableInputs* : *LL1PerformOperation*!(*input1*)!1
BY $\langle 3 \rangle 2$ DEF *LL1PerformOperation*

To simplify the writing of the proof, we re-state the definitions from the *LL1PerformOperation* action.

$\langle 4 \rangle$ *stateHash* $\triangleq$ *Hash*(*LL1RAM.publicState*, *LL1RAM.privateStateEnc*)
$\langle 4 \rangle$ *historyStateBinding* $\triangleq$ *Hash*(*LL1RAM.historySummary*, *stateHash*)
$\langle 4 \rangle$ *privateState* $\triangleq$ *SymmetricDecrypt*(*LL1NVRAM.symmetricKey*, *LL1RAM.privateStateEnc*)
$\langle 4 \rangle$ *sResult* $\triangleq$ *Service*(*LL1RAM.publicState*, *privateState*, *input1*)
$\langle 4 \rangle$ *newPrivateStateEnc* $\triangleq$
SymmetricEncrypt(*LL1NVRAM.symmetricKey*, *sResult.newPrivateState*)
$\langle 4 \rangle$ *newHistorySummary* $\triangleq$ *Hash*(*LL1NVRAM.historySummary*, *input1*)
$\langle 4 \rangle$ *newStateHash* $\triangleq$ *Hash*(*sResult.newPublicState*, *newPrivateStateEnc*)
$\langle 4 \rangle$ *newHistoryStateBinding* $\triangleq$ *Hash*(*newHistorySummary*, *newStateHash*)
$\langle 4 \rangle$ *newAuthenticator* $\triangleq$ *GenerateMAC*(*LL1NVRAM.symmetricKey*, *newHistoryStateBinding*)

We then assert the type safety of these definitions, with the help of the *LL1PerformOperationDefsTypeSafeLemma*.

$\langle 4 \rangle 2.$ $\wedge$ *stateHash* $\in$ *HashType*
$\wedge$ *historyStateBinding* $\in$ *HashType*
$\wedge$ *privateState* $\in$ *PrivateStateType*
$\wedge$ *sResult* $\in$ *ServiceResultType*
$\wedge$ *sResult.newPublicState* $\in$ *PublicStateType*
$\wedge$ *sResult.newPrivateState* $\in$ *PrivateStateType*
$\wedge$ *sResult.output* $\in$ *OutputType*
$\wedge$ *newPrivateStateEnc* $\in$ *PrivateStateEncType*
$\wedge$ *newHistorySummary* $\in$ *HashType*
$\wedge$ *newStateHash* $\in$ *HashType*
$\wedge$ *newHistoryStateBinding* $\in$ *HashType*

$\wedge$ *newAuthenticator* $\in$ *MACType*

⟨5⟩1. *input*1 $\in$ *LL*1*AvailableInputs*

BY ⟨4⟩1

⟨5⟩2. *LL*1*TypeInvariant*

BY ⟨2⟩1

⟨5⟩3. QED

BY ⟨5⟩1, ⟨5⟩2, *LL*1*PerformOperationDefsTypeSafeLemma*

We hide the definitions, so they don't overwhelm the prover. We'll pull them in as necessary below.

⟨4⟩ HIDE DEF *stateHash*, *historyStateBinding*, *privateState*, *sResult*, *newPrivateStateEnc*, *newHistorySummary*, *newStateHash*, *newHistoryStateBinding*, *newAuthenticator*

We prove some simple type statements up front, to avoid needing to repeat these multiple times below.

⟨4⟩3. $\wedge$ *LL*1*NVRAM*.*historySummary* $\in$ *HashType*
$\wedge$ *LL*1*NVRAM*.*symmetricKey* $\in$ *SymmetricKeyType*

⟨5⟩1. *LL*1*TypeInvariant*

BY ⟨2⟩1

⟨5⟩2. QED

BY ⟨5⟩1, *LL*1*SubtypeImplicationLemma*DEF *LL*1*SubtypeImplication*

⟨4⟩4. $\wedge$ *LL*1*NVRAM*.*historySummary*′ $\in$ *HashType*
$\wedge$ *LL*1*NVRAM*.*symmetricKey*′ $\in$ *SymmetricKeyType*

⟨5⟩1. *LL*1*TypeInvariant*′

BY ⟨2⟩1

⟨5⟩2. QED

BY ⟨5⟩1, *LL*1*SubtypeImplicationLemma*DEF *LL*1*SubtypeImplication*

We also prove that *LL1NVRAMHistorySummaryUncorrupted* predicate is true. This follows from the enablement conditions for *LL*1*PerformOperation*.

⟨4⟩5. *LL*1*NVRAMHistorySummaryUncorrupted*

We prove this directly from the definition of the *LL1NVRAMHistorySummaryUncorrupted* predicate. There is one type assumption and one antecedent in the implication.

⟨5⟩1. *stateHash* $\in$ *HashType*

BY ⟨4⟩2

⟨5⟩2. *LL*1*HistoryStateBindingAuthenticated*(*historyStateBinding*)

The authenticator in the RAM is a witness, because *TMPPerformOperation* ensures that it validates the history state binding.

⟨6⟩1. *ValidateMAC*(
*LL*1*NVRAM*.*symmetricKey*, *historyStateBinding*, *LL*1*RAM*.*authenticator*)

BY ⟨4⟩1 DEF *historyStateBinding*, *stateHash*

The *UnforgeabilityInvariant* then ensures that this authenticator is in the set of observed authenticators.

⟨6⟩2. *LL*1*RAM*.*authenticator* $\in$ *LL*1*ObservedAuthenticators*

⟨7⟩1. *UnforgeabilityInvariant*

BY ⟨2⟩1

⟨7⟩2. *historyStateBinding* $\in$ *HashType*

BY ⟨4⟩2

⟨7⟩3. QED

BY ⟨6⟩1, ⟨7⟩1, ⟨7⟩2 DEF *UnforgeabilityInvariant*, *LL*1*HistoryStateBindingAuthenticated*

These two conditions satisfy the definition of *LL*1*HistoryStateBindingAuthenticated*.

⟨6⟩3. QED

BY ⟨6⟩1, ⟨6⟩2 DEF *LL*1*HistoryStateBindingAuthenticated*

⟨5⟩3. *LL*1*NVRAM*.*historySummary* = *LL*1*RAM*.*historySummary*

BY ⟨4⟩1

⟨5⟩4. QED

BY ⟨5⟩1, ⟨5⟩2, ⟨5⟩3 DEF *LL*1*NVRAMHistorySummaryUncorrupted*, *historyStateBinding*

We'll prove each of the three invariants separately. First, we'll prove that the *InclusionInvariant* holds in the primed state.

$\langle 4 \rangle 6.$ $\mathit{InclusionInvariant}'$

To prove the universally quantified expression, we take a new set of variables of the appropriate types. For the TAKE step to be meaningful to the prover, first we have to tell the prover to expand the definition of *InclusionInvariant*, so it will see the universally quantified expression therein.

$\langle 5 \rangle$ USE DEF $\mathit{InclusionInvariant}$

$\langle 5 \rangle 1.$ TAKE $\mathit{input2} \in \mathit{InputType},$
$\mathit{historySummary} \in \mathit{HashType},$
$\mathit{publicState} \in \mathit{PublicStateType},$
$\mathit{privateStateEnc} \in \mathit{PrivateStateEncType}$

To simplify the writing of the proof, we re-state the definitions from the *InclusionInvariant*.

$\langle 5 \rangle$ $\mathit{inclStateHash} \triangleq \mathit{Hash}(\mathit{publicState}, \mathit{privateStateEnc})$
$\langle 5 \rangle$ $\mathit{inclHistoryStateBinding} \triangleq \mathit{Hash}(\mathit{historySummary}, \mathit{inclStateHash})$
$\langle 5 \rangle$ $\mathit{inclPrivateState} \triangleq \mathit{SymmetricDecrypt}(\mathit{LL1NVRAM}.\mathit{symmetricKey}, \mathit{privateStateEnc})$
$\langle 5 \rangle$ $\mathit{inclSResult} \triangleq \mathit{Service}(\mathit{publicState}, \mathit{inclPrivateState}, \mathit{input2})$
$\langle 5 \rangle$ $\mathit{inclNewPrivateStateEnc} \triangleq$
$\quad \mathit{SymmetricEncrypt}(\mathit{LL1NVRAM}.\mathit{symmetricKey}, \mathit{inclSResult}.\mathit{newPrivateState})$
$\langle 5 \rangle$ $\mathit{inclNewStateHash} \triangleq \mathit{Hash}(\mathit{inclSResult}.\mathit{newPublicState}, \mathit{inclNewPrivateStateEnc})$
$\langle 5 \rangle$ $\mathit{inclNewHistoryStateBinding} \triangleq \mathit{Hash}(\mathit{LL1NVRAM}.\mathit{historySummary}, \mathit{inclNewStateHash})$

We then assert the type safety of these definitions, with the help of the *InclusionInvariantDefsTypeSafeLemma*.

$\langle 5 \rangle 2.$ $\land$ $\mathit{inclStateHash} \in \mathit{HashType}$
$\land$ $\mathit{inclHistoryStateBinding} \in \mathit{HashType}$
$\land$ $\mathit{inclPrivateState} \in \mathit{PrivateStateType}$
$\land$ $\mathit{inclSResult} \in \mathit{ServiceResultType}$
$\land$ $\mathit{inclSResult}.\mathit{newPublicState} \in \mathit{PublicStateType}$
$\land$ $\mathit{inclSResult}.\mathit{newPrivateState} \in \mathit{PrivateStateType}$
$\land$ $\mathit{inclSResult}.\mathit{output} \in \mathit{OutputType}$
$\land$ $\mathit{inclNewPrivateStateEnc} \in \mathit{PrivateStateEncType}$
$\land$ $\mathit{inclNewStateHash} \in \mathit{HashType}$
$\land$ $\mathit{inclNewHistoryStateBinding} \in \mathit{HashType}$

$\langle 6 \rangle 1.$ $\mathit{LL1TypeInvariant}$
BY $\langle 2 \rangle 1$

$\langle 6 \rangle 2.$ QED
BY $\langle 5 \rangle 1$, $\langle 6 \rangle 1$, $\mathit{InclusionInvariantDefsTypeSafeLemma}$

The *InclusionInvariant* states an implication. To prove this, it suffices to assume the antecedent and prove the consequent.

$\langle 5 \rangle 3.$ SUFFICES
ASSUME
$\land$ $\mathit{LL1NVRAM}.\mathit{historySummary}' = \mathit{Hash}(\mathit{historySummary}, \mathit{input2})$
$\land$ $\mathit{LL1HistoryStateBindingAuthenticated}(\mathit{inclHistoryStateBinding})'$
PROVE
$\land$ $\mathit{inclSResult}.\mathit{output}' \in \mathit{LL1ObservedOutputs}'$
$\land$ $\mathit{LL1HistoryStateBindingAuthenticated}(\mathit{inclNewHistoryStateBinding})'$
OBVIOUS

We hide the definition of *InclusionInvariant* and the definitions from the *InclusionInvariant*.

$\langle 5 \rangle$ HIDE DEF $\mathit{InclusionInvariant}$
$\langle 5 \rangle$ HIDE DEF $\mathit{inclStateHash}$, $\mathit{inclHistoryStateBinding}$, $\mathit{inclPrivateState}$, $\mathit{inclSResult}$,
$\mathit{inclNewPrivateStateEnc}$, $\mathit{inclNewStateHash}$, $\mathit{inclNewHistoryStateBinding}$

Starting the *InclusionInvariant* proof proper, there are four main steps.

First, we prove that the history summary in the *NVRAM* matches the history summary that satisfies the antecedent condition, and that the input supplied to the *LL1PerformOperation* action matches the input that satisfies the antecedent condition.

Second, we prove that the public and private state in the RAM matches the public and private state that satisfies the antecedent condition.

The third and fourth steps each use the above results to prove one of the conjuncts in the consequent of the *InclusionInvariant*.

$\langle 5 \rangle 4.\ \wedge\ LL1NVRAM.historySummary = historySummary$
$\quad \wedge\ input1 = input2$

To prove the above equivalences, we will use the *HashCollisionResistant* property. To use this, we first have to prove some simple types.

$\langle 6 \rangle 1.\ LL1NVRAM.historySummary \in HashDomain$
$\langle 7 \rangle 1.\ LL1NVRAM.historySummary \in HashType$
BY $\langle 4 \rangle 3$
$\langle 7 \rangle 2.$ QED
BY $\langle 7 \rangle 1$ DEF *HashDomain*
$\langle 6 \rangle 2.\ input1 \in HashDomain$
$\langle 7 \rangle 1.\ input1 \in InputType$
$\langle 8 \rangle 1.\ input1 \in LL1AvailableInputs$
BY $\langle 4 \rangle 1$
$\langle 8 \rangle 2.\ LL1AvailableInputs \subseteq InputType$
$\langle 9 \rangle 1.\ LL1TypeInvariant$
BY $\langle 2 \rangle 1$
$\langle 9 \rangle 2.$ QED
BY $\langle 9 \rangle 1$ DEF *LL1TypeInvariant*
$\langle 8 \rangle 3.$ QED
BY $\langle 8 \rangle 1$, $\langle 8 \rangle 2$
$\langle 7 \rangle 2.$ QED
BY $\langle 7 \rangle 1$ DEF *HashDomain*
$\langle 6 \rangle 3.\ historySummary \in HashDomain$
$\langle 7 \rangle 1.\ historySummary \in HashType$
BY $\langle 5 \rangle 1$
$\langle 7 \rangle 2.$ QED
BY $\langle 7 \rangle 1$ DEF *HashDomain*
$\langle 6 \rangle 4.\ input2 \in HashDomain$
$\langle 7 \rangle 1.\ input2 \in InputType$
BY $\langle 5 \rangle 1$
$\langle 7 \rangle 2.$ QED
BY $\langle 7 \rangle 1$ DEF *HashDomain*

Then, we have to prove that the hashes are equal, which follows from the definition of the new history summary produced by *LL1PerformOperation*.

$\langle 6 \rangle 5.\ Hash(LL1NVRAM.historySummary, input1) = Hash(historySummary, input2)$
$\langle 7 \rangle 1.\ newHistorySummary = Hash(LL1NVRAM.historySummary, input1)$
BY DEF *newHistorySummary*
$\langle 7 \rangle 2.\ LL1NVRAM.historySummary' = Hash(historySummary, input2)$
BY $\langle 5 \rangle 3$
$\langle 7 \rangle 3.\ LL1NVRAM.historySummary' = newHistorySummary$
$\langle 8 \rangle 1.\ LL1NVRAM' = [historySummary \mapsto newHistorySummary,$
$\quad symmetricKey \mapsto LL1NVRAM.symmetricKey]$
BY $\langle 4 \rangle 1$ DEF *newHistorySummary*, *newPrivateStateEnc*, *sResult*, *privateState*
$\langle 8 \rangle 2.$ QED

BY $\langle 8\rangle 1$
$\langle 7\rangle 4.$ QED
BY $\langle 7\rangle 1$, $\langle 7\rangle 2$, $\langle 7\rangle 3$
$\langle 6\rangle 6.$ QED

Ideally, this QED step should just read:

BY $\langle 6\rangle 1$, $\langle 6\rangle 2$, $\langle 6\rangle 3$, $\langle 6\rangle 4$, $\langle 6\rangle 5$, *HashCollisionResistant*

However, the prover seems to get a little confused in this instance. We make life easier for the prover by definining some local variables and hiding their definitions before appealing to the *HashCollisionResistant* assumption.

$\langle 7\rangle\ h1a \triangleq LL1NVRAM.historySummary$
$\langle 7\rangle\ h2a \triangleq input1$
$\langle 7\rangle\ h1b \triangleq historySummary$
$\langle 7\rangle\ h2b \triangleq input2$
$\langle 7\rangle 1.\ h1a \in HashDomain$
BY $\langle 6\rangle 1$
$\langle 7\rangle 2.\ h2a \in HashDomain$
BY $\langle 6\rangle 2$
$\langle 7\rangle 3.\ h1b \in HashDomain$
BY $\langle 6\rangle 3$
$\langle 7\rangle 4.\ h2b \in HashDomain$
BY $\langle 6\rangle 4$
$\langle 7\rangle 5.\ Hash(h1a, h2a) = Hash(h1b, h2b)$
BY $\langle 6\rangle 5$
$\langle 7\rangle 6.\ h1a = h1b \land h2a = h2b$
$\langle 8\rangle$ HIDE DEF $h1a, h2a, h1b, h2b$
$\langle 8\rangle 1.$ QED
BY $\langle 7\rangle 1$, $\langle 7\rangle 2$, $\langle 7\rangle 3$, $\langle 7\rangle 4$, $\langle 7\rangle 5$, *HashCollisionResistant*
$\langle 7\rangle 7.$ QED
BY $\langle 7\rangle 6$

The second main step in the *InclusionInvariant* proof is to prove that the public and private state in the RAM matches the public and private state that satisfies the antecedent condition.

$\langle 5\rangle 5. \land\ publicState = LL1RAM.publicState$
$\land\ inclPrivateState' = privateState$

Most of the work of proving this step is proving that the public state in the RAM matches the public state that satisfies the antecedent condition, and that the encrypted private state in the RAM matches the encrypted private state that satisfies the antecedent condition.

$\langle 6\rangle 1. \land publicState = LL1RAM.publicState$
$\land privateStateEnc = LL1RAM.privateStateEnc$

To prove the above equivalences, we will use the *HashCollisionResistant* property. To use this, we first have to prove some simple types.

$\langle 7\rangle 1.\ publicState \in HashDomain$
$\langle 8\rangle 1.\ publicState \in PublicStateType$
BY $\langle 2\rangle 1$ DEF $LL1Refinement$
$\langle 8\rangle 2.$ QED
BY $\langle 8\rangle 1$ DEF $HashDomain$
$\langle 7\rangle 2. \land\ LL1RAM.publicState \in HashDomain$
$\land\ LL1RAM.privateStateEnc \in HashDomain$
$\langle 8\rangle 1. \land\ LL1RAM.publicState \in PublicStateType$
$\land\ LL1RAM.privateStateEnc \in PrivateStateEncType$
$\langle 9\rangle 1.\ LL1TypeInvariant$
BY $\langle 2\rangle 1$

⟨9⟩2. QED

BY ⟨9⟩1, *LL1SubtypeImplicationLemma*DEF *LL1SubtypeImplication*

⟨8⟩2. QED

BY ⟨8⟩1 DEF *HashDomain*

⟨7⟩3. $privateStateEnc \in HashDomain$

⟨8⟩1. $privateStateEnc \in PrivateStateEncType$

BY ⟨5⟩1

⟨8⟩2. QED

BY ⟨8⟩1 DEF *HashDomain*

Then, we'll show that the state hash defined in the *InclusionInvariant* matches the state hash defined in *LL1PerformOperation*. This is the hash equivalence we will use in our appeal to the *HashCollisionResistant* property.

This next step is quite involved, so the remainder of this proof follows much further down.

⟨7⟩4. $inclStateHash' = stateHash$

We can show that the state hashes match using the *UniquenessInvariant*. This will require proving some simple type statements, immediately below, and then proving that the two history state bindings corresponding to the two hashes are both authenticated.

⟨8⟩1. $inclStateHash' \in HashType$

⟨9⟩1. $publicState \in HashDomain$

⟨10⟩1. $publicState \in PublicStateType$

BY ⟨5⟩1

⟨10⟩2. QED

BY ⟨10⟩1 DEF *HashDomain*

⟨9⟩2. $privateStateEnc \in HashDomain$

⟨10⟩1. $privateStateEnc \in PrivateStateEncType$

BY ⟨5⟩1

⟨10⟩2. QED

BY ⟨10⟩1 DEF *HashDomain*

⟨9⟩3. QED

BY ⟨9⟩1, ⟨9⟩2, *HashTypeSafe*DEF *inclStateHash*

⟨8⟩2. $stateHash \in HashType$

BY ⟨4⟩2

The first history state binding we have to prove to be authenticated is the history state binding defined in the *InclusionInvariant*. This is by far the more involved of the two.

Our strategy is to first prove that this history state binding is authenticated in the primed state. Then, we prove that this history state binding is not validated by the new authenticator defined in the *LL1PerformOperation* action. Since this new authenticator is the only element that is in the primed set of observed authenticators but not in the unprimed set of observed authenticators, it follows that the the history state binding is authenticated by an authenticator in the unprimed set of authenticators, so it is authenticated in the unprimed state.

⟨8⟩3. $LL1HistoryStateBindingAuthenticated(inclHistoryStateBinding)$

The history state binding defined by the *InclusionInvariant* is authenticated in the primed state by hypothesis.

⟨9⟩1. $LL1HistoryStateBindingAuthenticated(inclHistoryStateBinding)'$

BY ⟨5⟩3

We have to prove that the history state binding defined by the *InclusionInvariant* is not authenticated by the new authenticator defined in the *LL1PerformOperation* action.

⟨9⟩2. $\neg ValidateMAC($
$LL1NVRAM.symmetricKey,$
$inclHistoryStateBinding,$
$newAuthenticator)$

Our strategy is to use proof by contradiction. First, we will show that if the history state binding defined by the *InclusionInvariant* were authenticated by the new authenticator defined in the *LL1PerformOperation* action, then this history state binding would equal the new history state binding defined in the *LL1PerformOperation* action. This then leads to the conclusion that the two history summaries in each of these history state bindings are equal. However, we will show that these two history summaries cannot be equal, because one was generated from a hash of the other.

⟨10⟩1. SUFFICES
ASSUME
$ValidateMAC($
$LL1NVRAM.symmetricKey,$
$inclHistoryStateBinding,$
$newAuthenticator)$
PROVE FALSE
OBVIOUS

In our contradictory universe, the history state binding defined by the *InclusionInvariant* equals the new history state binding defined in the *LL1PerformOperation* action, due to the collision resistance of the *MAC*. We merely need to prove some types, and then we can employ the *MACCollisionResistant* property directly.

⟨10⟩2. $inclHistoryStateBinding = newHistoryStateBinding$
⟨11⟩1. $LL1NVRAM.symmetricKey \in SymmetricKeyType$
BY ⟨4⟩3
⟨11⟩2. $inclHistoryStateBinding \in HashType$
BY ⟨5⟩2
⟨11⟩3. $newHistoryStateBinding \in HashType$
BY ⟨4⟩2
⟨11⟩4. QED
BY ⟨10⟩1, ⟨11⟩1, ⟨11⟩2, ⟨11⟩3, *MACCollisionResistant*
DEF *newAuthenticator*

In our contradictory universe, the history summary quantified by the *InclusionInvariant* equals the new history summary defined in the *LL1PerformOperation* action, due to the collision resistance of the hash function. We merely need to prove some types, and then we can employ the *HashCollisionResistant* property directly.

⟨10⟩3. $historySummary = newHistorySummary$
⟨11⟩1. $historySummary \in HashDomain$
⟨12⟩1. $historySummary \in HashType$
BY ⟨5⟩3
⟨12⟩2. QED
BY ⟨12⟩1 DEF *HashDomain*
⟨11⟩2. $inclStateHash \in HashDomain$
⟨12⟩1. $inclStateHash \in HashType$
BY ⟨5⟩2
⟨12⟩2. QED
BY ⟨12⟩1 DEF *HashDomain*
⟨11⟩3. $newHistorySummary \in HashDomain$
⟨12⟩1. $newHistorySummary \in HashType$
BY ⟨4⟩2
⟨12⟩2. QED
BY ⟨12⟩1 DEF *HashDomain*
⟨11⟩4. $newStateHash \in HashDomain$
⟨12⟩1. $newStateHash \in HashType$
BY ⟨4⟩2
⟨12⟩2. QED
BY ⟨12⟩1 DEF *HashDomain*

$\langle 11\rangle 5.$ QED
BY $\langle 10\rangle 2$, $\langle 11\rangle 1$, $\langle 11\rangle 2$, $\langle 11\rangle 3$, $\langle 11\rangle 4$, $HashCollisionResistant$
DEF $inclHistoryStateBinding$, $newHistoryStateBinding$

Back in the real universe, we will prove that the history summary quantified by the $InclusionInvariant$ cannot equal the new history summary defined in the $LL1PerformOperation$ action. The proof relies on the property of hash cardinality, though not on the $CardinalityInvariant$. Basically, we prove that the cardinality of the history summary in the new authenticator defined by $LL1PerformOperation$ is one greater than the hash cardinality of the history summary previously in the $NVRAM$. Since the hash cardinalities differ, it follows that the history summaries differ. The proof is tedious but straightforward.

$\langle 10\rangle 4.$ $historySummary \neq newHistorySummary$

First, we prove a bunch of types that are needed by the hash cardinality assumptions or for proving basic arithmetic.

$\langle 11\rangle 1.$ $input1 \in InputType$
$\langle 12\rangle 1.$ $input1 \in LL1AvailableInputs$
BY $\langle 4\rangle 1$
$\langle 12\rangle 2.$ $LL1AvailableInputs \subseteq InputType$
$\langle 13\rangle 1.$ $LL1TypeInvariant$
BY $\langle 2\rangle 1$
$\langle 13\rangle 2.$ QED
BY $\langle 13\rangle 1$ DEF $LL1TypeInvariant$
$\langle 12\rangle 3.$ QED
BY $\langle 12\rangle 1$, $\langle 12\rangle 2$
$\langle 11\rangle 2.$ $input1 \in HashDomain$
BY $\langle 11\rangle 1$ DEF $HashDomain$
$\langle 11\rangle 3.$ $LL1NVRAM.historySummary \in HashDomain$
$\langle 12\rangle 1.$ $LL1NVRAM.historySummary \in HashType$
BY $\langle 4\rangle 3$
$\langle 12\rangle 2.$ QED
BY $\langle 12\rangle 1$ DEF $HashDomain$
$\langle 11\rangle 4.$ $HashCardinality(input1) \in Nat$
BY $\langle 11\rangle 2$, $HashCardinalityTypeSafe$
$\langle 11\rangle 5.$ $HashCardinality(newHistorySummary) \in Nat$
$\langle 12\rangle 1.$ $newHistorySummary \in HashDomain$
$\langle 13\rangle 1.$ $newHistorySummary \in HashType$
BY $\langle 4\rangle 2$
$\langle 13\rangle 2.$ QED
BY $\langle 13\rangle 1$ DEF $HashDomain$
$\langle 12\rangle 2.$ QED
BY $\langle 12\rangle 1$, $HashCardinalityTypeSafe$
$\langle 11\rangle 6.$ $HashCardinality(historySummary) \in Nat$
$\langle 12\rangle 1.$ $historySummary \in HashDomain$
$\langle 13\rangle 1.$ $historySummary \in HashType$
BY $\langle 5\rangle 1$
$\langle 13\rangle 2.$ QED
BY $\langle 13\rangle 1$ DEF $HashDomain$
$\langle 12\rangle 2.$ QED
BY $\langle 12\rangle 1$, $HashCardinalityTypeSafe$

With the type statements out of the way, we can construct a simple inequality. We do this in four linear steps.

$\langle 11\rangle 7.$ $HashCardinality(newHistorySummary) =$
$HashCardinality(LL1NVRAM.historySummary) + HashCardinality(input1) + 1$

BY $\langle 11\rangle 2$, $\langle 11\rangle 3$, *HashCardinalityAccumulative* DEF *newHistorySummary*

$\langle 11\rangle 8$. $HashCardinality(newHistorySummary) = HashCardinality(historySummary) + HashCardinality(input1) + 1$

$\langle 12\rangle 1$. $LL1NVRAM.historySummary = historySummary$

BY $\langle 5\rangle 4$

$\langle 12\rangle 2$. QED

BY $\langle 11\rangle 7$, $\langle 12\rangle 1$

$\langle 11\rangle 9$. $HashCardinality(newHistorySummary) = HashCardinality(historySummary) + 1$

$\langle 12\rangle 1$. $HashCardinality(input1) = 0$

BY $\langle 11\rangle 1$, *InputCardinalityZero*

$\langle 12\rangle 2$. QED

BY $\langle 11\rangle 5$, $\langle 11\rangle 6$, $\langle 11\rangle 8$, $\langle 12\rangle 1$

$\langle 11\rangle 10$. $HashCardinality(newHistorySummary) \neq HashCardinality(historySummary)$

BY $\langle 11\rangle 5$, $\langle 11\rangle 6$, $\langle 11\rangle 9$

Since the hash cardinalities differ, it follows that the history summaries differ.

$\langle 11\rangle 11$. QED

BY $\langle 11\rangle 10$

The required contradiction follows from the previous two steps.

$\langle 10\rangle 5$. QED

BY $\langle 10\rangle 3$, $\langle 10\rangle 4$

Before completing the proof, we need to establish that the symmetric key in the *NVRAM* has not changed, because the *LL1HistoryStateBindingAuthenticated* predicate implicitly refers to this variable.

$\langle 9\rangle 3$. UNCHANGED $LL1NVRAM.symmetricKey$

BY $\langle 2\rangle 1$, *SymmetricKeyConstantLemma*

We also need to establish that the new authenticator defined by the *LL1PerformOperation* action is the only element that is in the primed set of observed authenticators but not in the unprimed set of observed authenticators.

$\langle 9\rangle 4$. $LL1ObservedAuthenticators' = LL1ObservedAuthenticators \cup \{newAuthenticator\}$

BY $\langle 4\rangle 1$ DEF *newAuthenticator*, *newHistoryStateBinding*, *newStateHash*, *newHistorySummary*, *newPrivateStateEnc*, *sResult*, *privateState*

The conclusion follows directly.

$\langle 9\rangle 5$. QED

BY $\langle 9\rangle 1$, $\langle 9\rangle 2$, $\langle 9\rangle 3$, $\langle 9\rangle 4$ DEF *LL1HistoryStateBindingAuthenticated*

The second history state binding we have to prove to be authenticated is the history state binding defined in the *LL1PerformOperation* action. We need to show that it is authenticated by some authenticator in the set of observed authenticators.

$\langle 8\rangle 4$. $LL1HistoryStateBindingAuthenticated(historyStateBinding)$

The authenticator in the RAM is a witness, because *TMPPerformOperation* ensures that it validates the history state binding.

$\langle 9\rangle 1$. $ValidateMAC(LL1NVRAM.symmetricKey, historyStateBinding, LL1RAM.authenticator)$

BY $\langle 4\rangle 1$ DEF *historyStateBinding*, *stateHash*

The *UnforgeabilityInvariant* then ensures that this authenticator is in the set of observed authenticators.

$\langle 9\rangle 2$. $LL1RAM.authenticator \in LL1ObservedAuthenticators$

$\langle 10\rangle 1$. *UnforgeabilityInvariant*

BY $\langle 2\rangle 1$

$\langle 10\rangle 2$. $historyStateBinding \in HashType$

BY $\langle 4\rangle 2$

$\langle 10\rangle 3$. QED

BY $\langle 9\rangle 1$, $\langle 10\rangle 1$, $\langle 10\rangle 2$ DEF *UnforgeabilityInvariant*, *LL1HistoryStateBindingAuthenticated*

These two conditions satisfy the definition of *LL1HistoryStateBindingAuthenticated*.

⟨9⟩3. QED
BY ⟨9⟩1, ⟨9⟩2 DEF *LL1HistoryStateBindingAuthenticated*

The conclusion follows fairly directly from the *UniquenessInvariant*. However, one small hitch is that the *UniquenessInvariant* requires both history state bindings to be defined using *LL1NVRAM.historySummary*, but *inclHistoryStateBinding* is defined using *historySummary* rather than *LL1NVRAM.historySummary*. Therefore, we have to add in the fact that *LL1NVRAM.historySummary = historySummary*.

⟨8⟩5. QED
⟨9⟩1. *UniquenessInvariant*
BY ⟨2⟩1
⟨9⟩2. *LL1NVRAM.historySummary = historySummary*
BY ⟨5⟩4
⟨9⟩3. *LL1NVRAM.historySummary = LL1RAM.historySummary*
BY ⟨4⟩1
⟨9⟩4. QED
BY ⟨8⟩1, ⟨8⟩2, ⟨8⟩3, ⟨8⟩4, ⟨9⟩1, ⟨9⟩2, ⟨9⟩3
DEF *UniquenessInvariant*, *inclHistoryStateBinding*, *historyStateBinding*

The conclusion follows directly from applying the *HashCollisionResistant* property.

⟨7⟩5. QED
BY ⟨7⟩1, ⟨7⟩2, ⟨7⟩3, ⟨7⟩4, *HashCollisionResistant*DEF *inclStateHash*, *stateHash*

The remainder of this step follows trivially.

⟨6⟩2. *publicState = LL1RAM.publicState*
BY ⟨6⟩1
⟨6⟩3. $inclPrivateState' = privateState$
⟨7⟩1. *privateStateEnc = LL1RAM.privateStateEnc*
BY ⟨6⟩1
⟨7⟩2. UNCHANGED *LL1NVRAM.symmetricKey*
BY ⟨2⟩1, *SymmetricKeyConstantLemma*
⟨7⟩3. QED
BY ⟨7⟩1, ⟨7⟩2 DEF *inclPrivateState*, *privateState*
⟨6⟩4. QED
BY ⟨6⟩2, ⟨6⟩3

The third main step in the *InclusionInvariant* proof is to prove the first conjunct in the consequent of the *InclusionInvariant*, namely that the primed output of the service is in the primed set of observed outputs.

This follows fairly directly from the just-proven facts that the variables that satisfy the antecedent conditions match the corresponding values in the *NVRAM* and the input value provided to the *LL1PerformOperation* action.

⟨5⟩6. $inclSResult'.output \in LL1ObservedOutputs'$
⟨6⟩1. $LL1ObservedOutputs' = LL1ObservedOutputs \cup \{inclSResult'.output\}$
⟨7⟩1. $LL1ObservedOutputs' = LL1ObservedOutputs \cup \{sResult.output\}$
BY ⟨4⟩1 DEF *sResult*, *privateState*
⟨7⟩2. $inclSResult'.output = sResult.output$
⟨8⟩1. $inclSResult' = sResult$
⟨9⟩1. $\land\ publicState = LL1RAM.publicState$
$\land\ inclPrivateState' = privateState$
BY ⟨5⟩5
⟨9⟩2. $input2 = input1$
BY ⟨5⟩4
⟨9⟩3. QED
BY ⟨9⟩1, ⟨9⟩2 DEF *inclSResult*, *sResult*
⟨8⟩2. QED

BY $\langle 8 \rangle 1$

$\langle 7 \rangle 3.$ QED

BY $\langle 7 \rangle 1$, $\langle 7 \rangle 2$

$\langle 6 \rangle 2.$ QED

BY $\langle 6 \rangle 1$

The fourth main step in the *InclusionInvariant* proof is to prove the second conjunct in the consequent of the *InclusionInvariant*, namely that the new history state binding defined in the *InclusionInvariant* is authenticated in the primed state.

We need to show that there exists some authenticator in the primed set of observed authenticators that authenticates this history state binding. Our witness is the new authenticator defined by the *LL1PerformOperation* action.

$\langle 5 \rangle 7.$ $LL1HistoryStateBindingAuthenticated(inclNewHistoryStateBinding)'$

The new authenticator defined in the *LL1PerformOperation* action is unioned into the set of observed authenticators by the *LL1PerformOperation* action.

$\langle 6 \rangle 1.$ $newAuthenticator \in LL1ObservedAuthenticators'$

$\langle 7 \rangle 1.$ $LL1ObservedAuthenticators' =$
$\quad LL1ObservedAuthenticators \cup \{newAuthenticator\}$

BY $\langle 4 \rangle 1$ DEF $newAuthenticator$, $newHistoryStateBinding$, $newStateHash$, $newHistorySummary$, $newPrivateStateEnc$, $sResult$, $privateState$

$\langle 7 \rangle 2.$ QED

BY $\langle 7 \rangle 1$

To prove that the new authenticator defined by the *LL1PerformOperation* action authenticates the new history state binding defined in the *InclusionInvariant*, we show that this authenticator was generated using this history state binding and using the same key.

$\langle 6 \rangle 2.$ $ValidateMAC(LL1NVRAM.symmetricKey', inclNewHistoryStateBinding', newAuthenticator)$

First, we show that the new history state binding defined by the *LL1PerformOperation* action is equal to the primed state of the new history state binding defined in the *InclusionInvariant*.

$\langle 7 \rangle 1.$ $inclNewHistoryStateBinding' = newHistoryStateBinding$

The proof is fairly straightforward. Using the above-proven facts that the variables that satisfy the antecedent conditions match the corresponding values in the *NVRAM* and the input value provided to the *LL1PerformOperation* action, we show that the results of the service in the primed state of the *InclusionInvariant* are the same as the results of the service in the *LL1PerformOperation* action. From there, we show that the state hashes are equal and that the history summaries are equal, which together imply that the state bindings are equal.

$\langle 8 \rangle 1.$ UNCHANGED $LL1NVRAM.symmetricKey$

BY $\langle 2 \rangle 1$, $SymmetricKeyConstantLemma$

$\langle 8 \rangle 2.$ $inclSResult' = sResult$

$\langle 9 \rangle 1.$ $\land\ publicState = LL1RAM.publicState$
$\quad \land\ inclPrivateState' = privateState$

BY $\langle 5 \rangle 5$

$\langle 9 \rangle 2.$ $input2 = input1$

BY $\langle 5 \rangle 4$

$\langle 9 \rangle 3.$ QED

BY $\langle 9 \rangle 1$, $\langle 9 \rangle 2$ DEF $inclSResult$, $sResult$

$\langle 8 \rangle 3.$ $inclSResult'.newPublicState = sResult.newPublicState$

BY $\langle 8 \rangle 2$

$\langle 8 \rangle 4.$ $inclSResult'.newPrivateState = sResult.newPrivateState$

BY $\langle 8 \rangle 2$

$\langle 8 \rangle 5.$ $inclNewPrivateStateEnc' = newPrivateStateEnc$

BY $\langle 8 \rangle 1$, $\langle 8 \rangle 4$ DEF $inclNewPrivateStateEnc$, $newPrivateStateEnc$

$\langle 8 \rangle 6.$ $inclNewStateHash' = newStateHash$

BY $\langle 8 \rangle 3$, $\langle 8 \rangle 5$ DEF $inclNewStateHash$, $newStateHash$

$\langle 8\rangle 7.\ \mathit{LL1NVRAM.historySummary}' = \mathit{newHistorySummary}$

$\langle 10\rangle 1.\ \mathit{LL1NVRAM}' = [\mathit{historySummary} \mapsto \mathit{newHistorySummary},$
$\qquad \mathit{symmetricKey} \mapsto \mathit{LL1NVRAM.symmetricKey}]$

BY $\langle 4\rangle 1$ DEF *newHistorySummary*, *newPrivateStateEnc*, *sResult*, *privateState*

$\langle 10\rangle 2.$ QED

BY $\langle 10\rangle 1$

$\langle 8\rangle 8.$ QED

BY $\langle 8\rangle 6$, $\langle 8\rangle 7$ DEF *inclNewHistoryStateBinding*, *newHistoryStateBinding*

Given that the state bindings are equal (and showing that the symmetric keys are equal, we can show that the new authenticator defined by the *LL1PerformOperation* action is equal to a *MAC* generated with the same inputs we are attempting to validate.

$\langle 7\rangle 2.\ \mathit{newAuthenticator} =$
$\qquad \mathit{GenerateMAC}(\mathit{LL1NVRAM.symmetricKey}', \mathit{inclNewHistoryStateBinding}')$

$\langle 8\rangle 1.$ UNCHANGED $\mathit{LL1NVRAM.symmetricKey}$

BY $\langle 2\rangle 1$, *SymmetricKeyConstantLemma*

$\langle 8\rangle 2.$ QED

BY $\langle 7\rangle 1$, $\langle 8\rangle 1$ DEF *newAuthenticator*

We can thus use the *MACComplete* property to show that the generated *MAC* validates appropriately. To do this, we first need to prove some types.

$\langle 7\rangle 3.\ \mathit{LL1NVRAM.symmetricKey}' \in \mathit{SymmetricKeyType}$

$\langle 8\rangle 1.\ \mathit{LL1TypeInvariant}'$

BY $\langle 2\rangle 1$

$\langle 8\rangle 2.$ QED

BY $\langle 8\rangle 1$, *LL1SubtypeImplicationLemma*DEF *LL1SubtypeImplication*

Since we have no lemma to prove the following type, we include its entire type proof here.

$\langle 7\rangle 4.\ \mathit{inclNewHistoryStateBinding}' \in \mathit{HashType}$

$\langle 8\rangle 1.\ \mathit{inclSResult}' \in \mathit{ServiceResultType}$

$\langle 9\rangle 1.\ \mathit{publicState} \in \mathit{PublicStateType}$

BY $\langle 5\rangle 1$

$\langle 9\rangle 2.\ \mathit{inclPrivateState}' \in \mathit{PrivateStateType}$

$\langle 10\rangle 1.\ \mathit{LL1NVRAM.symmetricKey}' \in \mathit{SymmetricKeyType}$

$\langle 11\rangle 1.\ \mathit{LL1TypeInvariant}'$

BY $\langle 2\rangle 1$

$\langle 11\rangle 2.$ QED

BY $\langle 11\rangle 1$, *LL1SubtypeImplicationLemma*DEF *LL1SubtypeImplication*

$\langle 10\rangle 2.\ \mathit{privateStateEnc}' \in \mathit{PrivateStateEncType}$

BY $\langle 5\rangle 1$

$\langle 10\rangle 3.$ QED

BY $\langle 10\rangle 1$, $\langle 10\rangle 2$, *SymmetricDecryptionTypeSafe*DEF *inclPrivateState*

$\langle 9\rangle 3.\ \mathit{input2} \in \mathit{InputType}$

BY $\langle 5\rangle 1$

$\langle 9\rangle 4.$ QED

BY $\langle 9\rangle 1$, $\langle 9\rangle 2$, $\langle 9\rangle 3$, *ServiceTypeSafe*DEF *inclSResult*

$\langle 8\rangle 2.\ \mathit{LL1NVRAM.historySummary}' \in \mathit{HashDomain}$

$\langle 9\rangle 1.\ \mathit{LL1NVRAM.historySummary}' \in \mathit{HashType}$

$\langle 10\rangle 1.\ \mathit{LL1TypeInvariant}'$

BY $\langle 2\rangle 1$

$\langle 10\rangle 2.$ QED

BY $\langle 10\rangle 1$, *LL1SubtypeImplicationLemma*DEF *LL1SubtypeImplication*

$\langle 9\rangle 2.$ QED

BY $\langle 9\rangle 1$ DEF *HashDomain*

$\langle 8\rangle 3.\ \mathit{inclNewStateHash}' \in \mathit{HashDomain}$

⟨9⟩1. $inclNewStateHash' \in HashType$

⟨10⟩1. $inclSResult.newPublicState' \in HashDomain$

⟨11⟩1. $inclSResult.newPublicState' \in PublicStateType$

BY ⟨8⟩1 DEF *ServiceResultType*

⟨11⟩2. QED

BY ⟨11⟩1 DEF *HashDomain*

⟨10⟩2. $inclNewPrivateStateEnc' \in HashDomain$

⟨11⟩1. $inclNewPrivateStateEnc' \in PrivateStateEncType$

⟨12⟩1. $LL1NVRAM.symmetricKey' \in SymmetricKeyType$

⟨13⟩1. $LL1TypeInvariant'$

BY ⟨2⟩1

⟨13⟩2. QED

BY ⟨13⟩1, *LL1SubtypeImplicationLemma*DEF *LL1SubtypeImplication*

⟨12⟩2. $inclSResult.newPrivateState' \in PrivateStateType$

BY ⟨8⟩1 DEF *ServiceResultType*

⟨12⟩3. QED

BY ⟨12⟩1, ⟨12⟩2, *SymmetricEncryptionTypeSafe*DEF *inclNewPrivateStateEnc*

⟨11⟩2. QED

BY ⟨11⟩1 DEF *HashDomain*

⟨10⟩3. QED

BY ⟨10⟩1, ⟨10⟩2, *HashTypeSafe*DEF *inclNewStateHash*

⟨9⟩2. QED

BY ⟨9⟩1 DEF *HashDomain*

⟨8⟩4. QED

BY ⟨8⟩2, ⟨8⟩3, *HashTypeSafe*DEF *inclNewHistoryStateBinding*

Then, we appeal to the *MACComplete* property in a straightforward way.

⟨7⟩5. QED

BY ⟨7⟩2, ⟨7⟩3, ⟨7⟩4, *MACComplete*

⟨6⟩3. QED

BY ⟨6⟩1, ⟨6⟩2 DEF *LL1HistoryStateBindingAuthenticated*

Since both conjuncts are true, the conjunction is true.

⟨5⟩8. QED

BY ⟨5⟩6, ⟨5⟩7

Second, we'll prove the *CardinalityInvariant* in the primed state.

⟨4⟩7. $CardinalityInvariant'$

To prove the universally quantified expression, we take a new set of variables of the appropriate types. For the TAKE step to be meaningful to the prover, first we have to tell the prover to expand the definition of *CardinalityInvariant*, so it will see the universally quantified expression therein.

⟨5⟩ USE DEF *CardinalityInvariant*

⟨5⟩1. TAKE $historySummary \in HashType$, $stateHash2 \in HashType$

To simplify the writing of the proof, we re-state the definition from the *CardinalityInvariant*.

⟨5⟩ $cardHistoryStateBinding \triangleq Hash(historySummary, stateHash2)$

We then assert the type safety of these definitions, with the help of the *CardinalityInvariantDefsTypeSafeLemma*.

⟨5⟩2. $cardHistoryStateBinding \in HashType$

BY ⟨5⟩1, *CardinalityInvariantDefsTypeSafeLemma*

The *CardinalityInvariant* states an implication. To prove this, it suffices to assume the antecedent and prove the consequent.

⟨5⟩3. SUFFICES

ASSUME

$\wedge$ $LL1NVRAMHistorySummaryUncorrupted'$

$\wedge$ $LL1HistoryStateBindingAuthenticated(cardHistoryStateBinding)'$

PROVE
$HashCardinality(historySummary) \leq HashCardinality(LL1NVRAM.historySummary')$
OBVIOUS

We then hide the definitions.

⟨5⟩ HIDE DEF *CardinalityInvariant*
⟨5⟩ HIDE DEF *cardHistoryStateBinding*

First, we'll prove that the hash cardinality of the history summary in the *NVRAM* does not decrease when a *LL1PerformOperation* action occurs. This follows from the fact that the hash cardinality of the history summary in the *LL1NVRAM* increases by one when a *LL1PerformOperation* action occurs. This is straightforward though somewhat tedious.

⟨5⟩4. $HashCardinality(LL1NVRAM.historySummary) \leq$
$HashCardinality(LL1NVRAM.historySummary')$

First, we prove a bunch of types that are needed by the hash cardinality assumptions or for proving basic arithmetic.

⟨6⟩1. $input1 \in InputType$
⟨7⟩1. $input1 \in LL1AvailableInputs$
BY ⟨4⟩1
⟨7⟩2. $LL1AvailableInputs \subseteq InputType$
⟨8⟩1. *LL1TypeInvariant*
BY ⟨2⟩1
⟨8⟩2. QED
BY ⟨8⟩1 DEF *LL1TypeInvariant*
⟨7⟩3. QED
BY ⟨7⟩1, ⟨7⟩2
⟨6⟩2. $LL1NVRAM.historySummary \in HashDomain$
⟨7⟩1. $LL1NVRAM.historySummary \in HashType$
BY ⟨4⟩3
⟨7⟩2. QED
BY ⟨7⟩1 DEF *HashDomain*
⟨6⟩3. $LL1NVRAM.historySummary' \in HashDomain$
⟨7⟩1. $LL1NVRAM.historySummary' \in HashType$
BY ⟨4⟩4
⟨7⟩2. QED
BY ⟨7⟩1 DEF *HashDomain*
⟨6⟩4. $input1 \in HashDomain$
BY ⟨6⟩1 DEF *HashDomain*
⟨6⟩5. $HashCardinality(LL1NVRAM.historySummary) \in Nat$
BY ⟨6⟩2, *HashCardinalityTypeSafe*DEF *HashDomain*
⟨6⟩6. $HashCardinality(LL1NVRAM.historySummary') \in Nat$
BY ⟨6⟩3, *HashCardinalityTypeSafe*DEF *HashDomain*

With the type statements out of the way, we can construct a simple inequality. We do this in four linear steps.

⟨6⟩7. $HashCardinality(Hash(LL1NVRAM.historySummary, input1)) =$
$HashCardinality(LL1NVRAM.historySummary) + HashCardinality(input1) + 1$
BY ⟨6⟩2, ⟨6⟩4, *HashCardinalityAccumulative*
⟨6⟩8. $HashCardinality(LL1NVRAM.historySummary') =$
$HashCardinality(LL1NVRAM.historySummary) + HashCardinality(input1) + 1$
⟨7⟩1. $LL1NVRAM.historySummary' = Hash(LL1NVRAM.historySummary, input1)$
⟨8⟩1. $newHistorySummary = Hash(LL1NVRAM.historySummary, input1)$
BY DEF *newHistorySummary*
⟨8⟩2. $LL1NVRAM.historySummary' = newHistorySummary$
⟨9⟩1. $LL1NVRAM' = [$

$historySummary \mapsto newHistorySummary,$
$symmetricKey \mapsto LL1NVRAM.symmetricKey]$

BY ⟨4⟩1 DEF $newHistorySummary$, $newPrivateStateEnc$, $sResult$, $privateState$

⟨9⟩2. QED
BY ⟨9⟩1

⟨8⟩3. QED
BY ⟨8⟩1, ⟨8⟩2

⟨7⟩2. QED
BY ⟨7⟩1, ⟨6⟩7

⟨6⟩9. $HashCardinality(LL1NVRAM.historySummary') =$
$HashCardinality(LL1NVRAM.historySummary) + 1$

⟨7⟩3. $HashCardinality(input1) = 0$
BY ⟨6⟩1, $InputCardinalityZero$

⟨7⟩4. QED
BY ⟨6⟩5, ⟨6⟩6, ⟨6⟩8, ⟨7⟩3

⟨6⟩10. QED
BY ⟨6⟩5, ⟨6⟩6, ⟨6⟩9

Next, we'll prove that either the history state binding is authenticated in the unprimed state or the history state binding is authenticated by the new authenticator defined by the *LL1PerformOperation* action. This follows from the fact that the primed set of authenticators is constructed as the union of the unprimed set and the new authenticator.

⟨5⟩5. $\lor$ $LL1HistoryStateBindingAuthenticated(cardHistoryStateBinding)$
$\lor$ $ValidateMAC(LL1NVRAM.symmetricKey, cardHistoryStateBinding, newAuthenticator)$

⟨6⟩1. UNCHANGED $LL1NVRAM.symmetricKey$
BY ⟨2⟩1, $SymmetricKeyConstantLemma$

⟨6⟩2. $LL1ObservedAuthenticators' =$
$LL1ObservedAuthenticators \cup \{newAuthenticator\}$

BY ⟨4⟩1 DEF $newAuthenticator$, $newHistoryStateBinding$, $newStateHash$,
$newHistorySummary$, $newPrivateStateEnc$, $sResult$, $privateState$

⟨6⟩3. $LL1HistoryStateBindingAuthenticated(cardHistoryStateBinding)'$
BY ⟨5⟩3

⟨6⟩4. QED
BY ⟨6⟩1, ⟨6⟩2, ⟨6⟩3 DEF $LL1HistoryStateBindingAuthenticated$

Given the above disjunction, we proceed via case analysis. First, we consider the case in which the history state binding is authenticated in the unprimed state.

In this case, because the *CardinalityInvariant* is true in the unprimed state, we can prove that the hash cardinality of the history summary that satisfies the antecedent is less than or equal to the hash cardinality of the history summary in the unprimed *NVRAM*. Since the less-than-or-equal-to relation is transitive, this leads directly to our proof goal.

⟨5⟩6. CASE $LL1HistoryStateBindingAuthenticated(cardHistoryStateBinding)$

⟨6⟩2. $HashCardinality(historySummary) \leq HashCardinality(LL1NVRAM.historySummary)$

We know the *CardinalityInvariant* to be true in the unprimed state.

⟨7⟩1. $CardinalityInvariant$
BY ⟨2⟩1

The *CardinalityInvariant* has two type assumptions.

⟨7⟩2. $historySummary \in HashType$
BY ⟨5⟩1

⟨7⟩3. $stateHash2 \in HashType$
BY ⟨5⟩1

The implication in the *CardinalityInvariant* has antecedents. One is equal to the present case condition. The other is that the *LL1NVRAMHistorySummaryUncorrupted* predicate is true.

$\langle 7\rangle 4.$ *LL1NVRAMHistorySummaryUncorrupted*
BY $\langle 4\rangle 5$

The consequent of the implication in the *CardinalityInvariant* follows directly.

$\langle 7\rangle 5.$ QED
BY $\langle 5\rangle 6$, $\langle 7\rangle 1$, $\langle 7\rangle 2$, $\langle 7\rangle 3$, $\langle 7\rangle 4$
DEF *CardinalityInvariant*, *cardHistoryStateBinding*
$\langle 6\rangle 3.$ $HashCardinality(historySummary) \in Nat$
$\langle 7\rangle 1.$ $historySummary \in HashDomain$
$\langle 8\rangle 1.$ $historySummary \in HashType$
BY $\langle 5\rangle 1$
$\langle 8\rangle 2.$ QED
BY $\langle 8\rangle 1$ DEF *HashDomain*
$\langle 7\rangle 2.$ QED
BY $\langle 7\rangle 1$, *HashCardinalityTypeSafe*
$\langle 6\rangle 4.$ $HashCardinality(LL1NVRAM.historySummary) \in Nat$
$\langle 7\rangle 1.$ $LL1NVRAM.historySummary \in HashDomain$
$\langle 8\rangle 1.$ $LL1NVRAM.historySummary \in HashType$
BY $\langle 4\rangle 3$
$\langle 8\rangle 2.$ QED
BY $\langle 8\rangle 1$ DEF *HashDomain*
$\langle 7\rangle 2.$ QED
BY $\langle 7\rangle 1$, *HashCardinalityTypeSafe*
$\langle 6\rangle 5.$ $HashCardinality(LL1NVRAM.historySummary') \in Nat$
$\langle 7\rangle 1.$ $LL1NVRAM.historySummary' \in HashDomain$
$\langle 8\rangle 1.$ $LL1NVRAM.historySummary' \in HashType$
BY $\langle 4\rangle 4$
$\langle 8\rangle 2.$ QED
BY $\langle 8\rangle 1$ DEF *HashDomain*
$\langle 7\rangle 2.$ QED
BY $\langle 7\rangle 1$, *HashCardinalityTypeSafe*
$\langle 6\rangle 6.$ QED

Ideally, this QED step should just read:

BY $\langle 5\rangle 3$, $\langle 6\rangle 2$, $\langle 6\rangle 3$, $\langle 6\rangle 4$, $\langle 6\rangle 5$, *LEQTransitive*

However, the prover seems to get a little confused in this instance. We make life easier for the prover by defining some local variables and hiding their definitions before appealing to the *LEQTransitive* assumption.

$\langle 7\rangle\ x \triangleq HashCardinality(historySummary)$
$\langle 7\rangle\ y \triangleq HashCardinality(LL1NVRAM.historySummary)$
$\langle 7\rangle\ z \triangleq HashCardinality(LL1NVRAM.historySummary')$
$\langle 7\rangle 1.$ $x \in Nat$
BY $\langle 6\rangle 3$
$\langle 7\rangle 2.$ $y \in Nat$
BY $\langle 6\rangle 4$
$\langle 7\rangle 3.$ $z \in Nat$
BY $\langle 6\rangle 5$
$\langle 7\rangle 4.$ $x \leq y$
BY $\langle 6\rangle 2$
$\langle 7\rangle 5.$ $y \leq z$
BY $\langle 5\rangle 4$
$\langle 7\rangle 6.$ $x \leq z$
$\langle 8\rangle$ HIDE DEF x, y, z

$\langle 8 \rangle 1.$ QED
BY $\langle 7 \rangle 1$, $\langle 7 \rangle 2$, $\langle 7 \rangle 3$, $\langle 7 \rangle 4$, $\langle 7 \rangle 5$, *LEQTransitive*
$\langle 7 \rangle 7.$ QED
BY $\langle 7 \rangle 6$

In the other case, the history state binding is authenticated by the new authenticator defined by the *LL1PerformOperation* action

$\langle 5 \rangle 7.$ CASE $ValidateMAC(LL1NVRAM.symmetricKey, cardHistoryStateBinding, newAuthenticator)$

Since the new authenticator authenticates the new history state binding defined in the *LL1PerformOperation* action, it follows that the history state bindings are equal, by the *MACCollisionResistant* property.

$\langle 6 \rangle 1.$ $cardHistoryStateBinding = newHistoryStateBinding$
$\langle 7 \rangle 1.$ $LL1NVRAM.symmetricKey \in SymmetricKeyType$
BY $\langle 4 \rangle 3$
$\langle 7 \rangle 3.$ $cardHistoryStateBinding \in HashType$
BY $\langle 5 \rangle 2$
$\langle 7 \rangle 4.$ $newHistoryStateBinding \in HashType$
BY $\langle 4 \rangle 2$
$\langle 7 \rangle 5.$ QED
BY $\langle 5 \rangle 7$, $\langle 7 \rangle 1$, $\langle 7 \rangle 3$, $\langle 7 \rangle 4$, *MACCollisionResistant*DEF *newAuthenticator*

Since the inputs-state bindings are equal, it follows that the history summaries are equal, by the *HashCollisionResistant* property and the fact that the history summary in the primed *NVRAM* equals the new history summary defined in the *LL1PerformOperation* action.

$\langle 6 \rangle 2.$ $historySummary = LL1NVRAM.historySummary'$
$\langle 7 \rangle 1.$ $historySummary = newHistorySummary$
$\langle 8 \rangle 1.$ $historySummary \in HashDomain$
$\langle 9 \rangle 1.$ $historySummary \in HashType$
BY $\langle 5 \rangle 5$
$\langle 9 \rangle 2.$ QED
BY $\langle 9 \rangle 1$ DEF *HashDomain*
$\langle 8 \rangle 2.$ $stateHash2 \in HashDomain$
$\langle 9 \rangle 1.$ $stateHash2 \in HashType$
BY $\langle 5 \rangle 1$
$\langle 9 \rangle 2.$ QED
BY $\langle 9 \rangle 1$ DEF *HashDomain*
$\langle 8 \rangle 3.$ $newHistorySummary \in HashDomain$
$\langle 9 \rangle 1.$ $newHistorySummary \in HashType$
BY $\langle 4 \rangle 2$
$\langle 9 \rangle 2.$ QED
BY $\langle 9 \rangle 1$ DEF *HashDomain*
$\langle 8 \rangle 4.$ $newStateHash \in HashDomain$
$\langle 9 \rangle 1.$ $newStateHash \in HashType$
BY $\langle 4 \rangle 2$
$\langle 9 \rangle 2.$ QED
BY $\langle 9 \rangle 1$ DEF *HashDomain*
$\langle 8 \rangle 5.$ QED
BY $\langle 6 \rangle 1$, $\langle 8 \rangle 1$, $\langle 8 \rangle 2$, $\langle 8 \rangle 3$, $\langle 8 \rangle 4$, *HashCollisionResistant*
DEF *cardHistoryStateBinding*, *newHistoryStateBinding*
$\langle 7 \rangle 2.$ $LL1NVRAM.historySummary' = newHistorySummary$
$\langle 8 \rangle 1.$ $LL1NVRAM' = [historySummary \mapsto newHistorySummary, symmetricKey \mapsto LL1NVRAM.symmetricKey]$
BY $\langle 4 \rangle 1$ DEF *newHistorySummary*, *newPrivateStateEnc*, *sResult*, *privateState*
$\langle 8 \rangle 2.$ QED
BY $\langle 8 \rangle 1$

⟨7⟩3. QED
BY ⟨7⟩1, ⟨7⟩2

Since the history summaries are equal, their hash cardinalities are equal.

⟨6⟩3. $HashCardinality(historySummary) = HashCardinality(LL1NVRAM.historySummary')$
BY ⟨6⟩2

We then have to prove that the hash cardinalities are natural numbers, to enable the prover to conclude that the equality above implies the less-than-or-equal we are trying to prove.

⟨6⟩4. $HashCardinality(historySummary) \in Nat$
⟨7⟩1. $historySummary \in HashType$
BY ⟨5⟩1
⟨7⟩2. $historySummary \in HashDomain$
BY ⟨7⟩1 DEF $HashDomain$
⟨7⟩3. QED
BY ⟨7⟩2, $HashCardinalityTypeSafe$
⟨6⟩5. $HashCardinality(LL1NVRAM.historySummary') \in Nat$
⟨7⟩1. $LL1NVRAM.historySummary' \in HashType$
BY ⟨4⟩4
⟨7⟩2. $LL1NVRAM.historySummary' \in HashDomain$
BY ⟨7⟩1 DEF $HashDomain$
⟨7⟩3. QED
BY ⟨7⟩2, $HashCardinalityTypeSafe$
⟨6⟩6. QED
BY ⟨6⟩3, ⟨6⟩4, ⟨6⟩5

The two cases are exhaustive, so the *CardinalityInvariant* is proven.

⟨5⟩8. QED
BY ⟨5⟩5, ⟨5⟩6, ⟨5⟩7

Third, we'll prove the *UniquenessInvariant* in the primed state.

⟨4⟩8. $UniquenessInvariant'$

To prove the universally quantified expression, we take a new set of variables of the appropriate types. For the TAKE step to be meaningful to the prover, first we have to tell the prover to expand the definition of *UniquenessInvariant*, so it will see the universally quantified expression therein.

⟨5⟩ USE DEF $UniquenessInvariant$
⟨5⟩2. TAKE $stateHash1, stateHash2 \in HashType$

To simplify the writing of the proof, we re-state the definitions from the *UniquenessInvariant*.

⟨5⟩ $uniqHistoryStateBinding1 \triangleq Hash(LL1NVRAM.historySummary, stateHash1)$
⟨5⟩ $uniqHistoryStateBinding2 \triangleq Hash(LL1NVRAM.historySummary, stateHash2)$

The *UniquenessInvariant* states an implication. To prove this, it suffices to assume the antecedent and prove the consequent.

⟨5⟩3. SUFFICES
ASSUME $\land\ LL1HistoryStateBindingAuthenticated(uniqHistoryStateBinding1)'$
$\land\ LL1HistoryStateBindingAuthenticated(uniqHistoryStateBinding2)'$
PROVE $stateHash1 = stateHash2$
OBVIOUS

We hide the definitions of *UniquenessInvariant* and the definitions from the *UniquenessInvariant*.

⟨5⟩ HIDE DEF $UniquenessInvariant$
⟨5⟩ HIDE DEF $uniqHistoryStateBinding1, uniqHistoryStateBinding2$

The proof of $UniquenessInvariant'$ employs *CardinalityInvariant*. To facilitate our extremely simple arithmetic, we begin by proving the types of a couple of critical hash cardinalities.

⟨5⟩4. $HashCardinality(LL1NVRAM.historySummary) \in Nat$
⟨6⟩1. $LL1NVRAM.historySummary \in HashDomain$

⟨7⟩1. $LL1NVRAM.historySummary \in HashType$
BY ⟨4⟩3
⟨7⟩2. QED
BY ⟨7⟩1 DEF $HashDomain$
⟨6⟩2. QED
BY ⟨6⟩1, $HashCardinalityTypeSafe$
⟨5⟩5. $HashCardinality(LL1NVRAM.historySummary') \in Nat$
⟨6⟩1. $LL1NVRAM.historySummary' \in HashDomain$
⟨7⟩1. $LL1NVRAM.historySummary' \in HashType$
BY ⟨4⟩4
⟨7⟩2. QED
BY ⟨7⟩1 DEF $HashDomain$
⟨6⟩2. QED
BY ⟨6⟩1, $HashCardinalityTypeSafe$

We'll prove that the hash cardinality of the history summary in the *LL1NVRAM* increases when a *LL1PerformOperation* action occurs. This follows from the fact that the hash cardinality of the history summary in the *NVRAM* increases by one when a *LL1PerformOperation* action occurs. This is straightforward though somewhat tedious.

⟨5⟩6. $HashCardinality(LL1NVRAM.historySummary) <$
$HashCardinality(LL1NVRAM.historySummary')$

First, we prove a bunch of types that are needed by the hash cardinality assumptions or for proving basic arithmetic.

⟨6⟩1. $LL1NVRAM.historySummary \in HashDomain$
⟨7⟩1. $LL1NVRAM.historySummary \in HashType$
BY ⟨4⟩3
⟨7⟩2. QED
BY ⟨7⟩1 DEF $HashDomain$
⟨6⟩2. $LL1NVRAM.historySummary' \in HashDomain$
⟨7⟩1. $LL1NVRAM.historySummary' \in HashType$
BY ⟨4⟩4
⟨7⟩2. QED
BY ⟨7⟩1 DEF $HashDomain$
⟨6⟩3. $input1 \in HashDomain$
⟨7⟩1. $input1 \in InputType$
⟨8⟩1. $input1 \in LL1AvailableInputs$
BY ⟨4⟩1
⟨8⟩2. $LL1AvailableInputs \subseteq InputType$
⟨9⟩1. $LL1TypeInvariant$
BY ⟨2⟩1
⟨9⟩2. QED
BY ⟨9⟩1 DEF $LL1TypeInvariant$
⟨8⟩3. QED
BY ⟨8⟩1, ⟨8⟩2
⟨7⟩2. QED
BY ⟨7⟩1 DEF $HashDomain$

With the type statements out of the way, we can construct a simple inequality. We do this in four linear steps.

⟨6⟩4. $HashCardinality(Hash(LL1NVRAM.historySummary, input1)) =$
$HashCardinality(LL1NVRAM.historySummary) + HashCardinality(input1) + 1$
BY ⟨6⟩1, ⟨6⟩3, $HashCardinalityAccumulative$
⟨6⟩5. $HashCardinality(LL1NVRAM.historySummary') =$
$HashCardinality(LL1NVRAM.historySummary) + HashCardinality(input1) + 1$

$\langle 7\rangle 1.\ LL1NVRAM.historySummary' = Hash(LL1NVRAM.historySummary, input1)$

$\langle 8\rangle 1.\ newHistorySummary = Hash(LL1NVRAM.historySummary, input1)$

BY DEF $newHistorySummary$

$\langle 8\rangle 2.\ LL1NVRAM.historySummary' = newHistorySummary$

$\langle 9\rangle 1.\ LL1NVRAM' = [historySummary \mapsto newHistorySummary, symmetricKey \mapsto LL1NVRAM.symmetricKey]$

BY $\langle 4\rangle 1$ DEF $newHistorySummary$, $newPrivateStateEnc$, $sResult$, $privateState$

$\langle 9\rangle 2.$ QED

BY $\langle 9\rangle 1$

$\langle 8\rangle 3.$ QED

BY $\langle 8\rangle 1$, $\langle 8\rangle 2$

$\langle 7\rangle 2.$ QED

BY $\langle 7\rangle 1$, $\langle 6\rangle 4$

$\langle 6\rangle 6.\ HashCardinality(LL1NVRAM.historySummary') = HashCardinality(LL1NVRAM.historySummary) + 1$

$\langle 7\rangle 3.\ HashCardinality(input1) = 0$

$\langle 8\rangle 1.\ input1 \in InputType$

$\langle 9\rangle 1.\ input1 \in LL1AvailableInputs$

BY $\langle 4\rangle 1$

$\langle 9\rangle 2.\ LL1AvailableInputs \subseteq InputType$

$\langle 10\rangle 1.\ LL1TypeInvariant$

BY $\langle 2\rangle 1$

$\langle 10\rangle 2.$ QED

BY $\langle 10\rangle 1$ DEF $LL1TypeInvariant$

$\langle 9\rangle 3.$ QED

BY $\langle 9\rangle 1$, $\langle 9\rangle 2$

$\langle 8\rangle 2.$ QED

BY $\langle 8\rangle 1$, $InputCardinalityZero$

$\langle 7\rangle 4.$ QED

BY $\langle 5\rangle 4$, $\langle 5\rangle 5$, $\langle 6\rangle 5$, $\langle 7\rangle 3$

$\langle 6\rangle 7.$ QED

BY $\langle 5\rangle 4$, $\langle 5\rangle 5$, $\langle 6\rangle 6$

Next, we'll prove that the primed history state binding 1 defined by the *UniquenessInvariant* is equal to the new state history state binding defined by the *LL1PerformOperation* action.

$\langle 5\rangle 7.\ uniqHistoryStateBinding1' = newHistoryStateBinding$

$\langle 6\rangle 1.\ ValidateMAC(LL1NVRAM.symmetricKey, uniqHistoryStateBinding1', newAuthenticator)$

We start by proving that either the history state binding is authenticated in the unprimed state or the history state binding is authenticated by the new authenticator defined by the *LL1PerformOperation* action. This follows from the fact that the primed set of authenticators is constructed as the union of the unprimed set and the new authenticator.

$\langle 7\rangle 1.\ \lor LL1HistoryStateBindingAuthenticated(uniqHistoryStateBinding1')$
$\quad \lor ValidateMAC(LL1NVRAM.symmetricKey, uniqHistoryStateBinding1', newAuthenticator)$

$\langle 8\rangle 1.\ LL1HistoryStateBindingAuthenticated(uniqHistoryStateBinding1)'$

BY $\langle 5\rangle 3$

$\langle 8\rangle 2.$ UNCHANGED $LL1NVRAM.symmetricKey$

BY $\langle 2\rangle 1$, $SymmetricKeyConstantLemma$

$\langle 8\rangle 3.\ LL1ObservedAuthenticators' = LL1ObservedAuthenticators \cup \{newAuthenticator\}$

BY $\langle 4\rangle 1$ DEF $newAuthenticator$, $newHistoryStateBinding$, $newStateHash$, $newHistorySummary$, $newPrivateStateEnc$, $sResult$, $privateState$

$\langle 8\rangle 4.$ QED

BY $\langle 8\rangle 1$, $\langle 8\rangle 2$, $\langle 8\rangle 3$ DEF $LL1HistoryStateBindingAuthenticated$

We then prove that the history state binding is not authenticated in the unprimed state. We use proof by contradiction. We show that if the history state binding were authenticated in the unprimed state, we could conclude a hash cardinality inequality that contradicts the hash cardinality inequality we proved above.

$\langle 7\rangle 2.$ $\neg LL1HistoryStateBindingAuthenticated(uniqHistoryStateBinding1')$

$\langle 8\rangle 1.$ SUFFICES
ASSUME $LL1HistoryStateBindingAuthenticated(uniqHistoryStateBinding1')$
PROVE FALSE
OBVIOUS

$\langle 8\rangle 2.$ $HashCardinality(LL1NVRAM.historySummary') \leq HashCardinality(LL1NVRAM.historySummary)$

$\langle 9\rangle 1.$ $CardinalityInvariant$
BY $\langle 2\rangle 1$

$\langle 9\rangle 2.$ $LL1NVRAM.historySummary' \in HashType$
BY $\langle 4\rangle 4$

$\langle 9\rangle 3.$ $stateHash1 \in HashType$
BY $\langle 5\rangle 2$

$\langle 9\rangle 4.$ $LL1NVRAMHistorySummaryUncorrupted$
BY $\langle 4\rangle 5$

$\langle 9\rangle 5.$ QED
BY $\langle 8\rangle 1$, $\langle 9\rangle 1$, $\langle 9\rangle 2$, $\langle 9\rangle 3$, $\langle 9\rangle 4$ DEF $CardinalityInvariant$, $uniqHistoryStateBinding1$

$\langle 8\rangle 3.$ QED
BY $\langle 5\rangle 4$, $\langle 5\rangle 5$, $\langle 5\rangle 6$, $\langle 8\rangle 2$, $GEQorLT$

$\langle 7\rangle 3.$ QED
BY $\langle 7\rangle 1$, $\langle 7\rangle 2$

Since the new authenticator authenticates the new history state binding defined in the $LL1PerformOperation$ action, it follows that the history state bindings are equal, by the $MACCollisionResistant$ property.

We first have to prove some basic types, then we can appeal to the $MACCollisionResistant$ property.

$\langle 6\rangle 2.$ $LL1NVRAM.symmetricKey \in SymmetricKeyType$
BY $\langle 4\rangle 3$

$\langle 6\rangle 3.$ $uniqHistoryStateBinding1' \in HashType$

$\langle 7\rangle 1.$ $LL1NVRAM.historySummary' \in HashDomain$

$\langle 8\rangle 1.$ $LL1NVRAM.historySummary' \in HashType$

$\langle 9\rangle 1.$ $LL1TypeInvariant'$
BY $\langle 2\rangle 1$

$\langle 9\rangle 2.$ QED
BY $\langle 9\rangle 1$, $LL1SubtypeImplicationLemma$DEF $LL1SubtypeImplication$

$\langle 8\rangle 2.$ QED
BY $\langle 8\rangle 1$ DEF $HashDomain$

$\langle 7\rangle 2.$ $stateHash1 \in HashDomain$

$\langle 8\rangle 1.$ $stateHash1 \in HashType$
BY $\langle 5\rangle 2$

$\langle 8\rangle 2.$ QED
BY $\langle 3\rangle 1$ DEF $HashDomain$

$\langle 7\rangle 3.$ QED
BY $\langle 7\rangle 1$, $\langle 7\rangle 2$, $HashTypeSafe$DEF $uniqHistoryStateBinding1$

$\langle 6\rangle 4.$ $newHistoryStateBinding \in HashType$
BY $\langle 4\rangle 2$

$\langle 6\rangle 5.$ QED
BY $\langle 6\rangle 1$, $\langle 6\rangle 2$, $\langle 6\rangle 3$, $\langle 6\rangle 4$, $MACCollisionResistant$DEF $newAuthenticator$

Next, we'll prove that the primed history state binding 2 defined by the $UniquenessInvariant$ is equal to the new state history state binding defined by the $LL1PerformOperation$ action.

$\langle 5\rangle 8.$ $uniqHistoryStateBinding2' = newHistoryStateBinding$

$\langle 6 \rangle 1.$ $ValidateMAC(LL1NVRAM.symmetricKey, uniqHistoryStateBinding2', newAuthenticator)$

We start by proving that either the history state binding is authenticated in the unprimed state or the history state binding is authenticated by the new authenticator defined by the *LL1PerformOperation* action. This follows from the fact that the primed set of authenticators is constructed as the union of the unprimed set and the new authenticator.

$\langle 7 \rangle 1.$ $\vee\ LL1HistoryStateBindingAuthenticated(uniqHistoryStateBinding2')$
$\quad\vee\ ValidateMAC(LL1NVRAM.symmetricKey, uniqHistoryStateBinding2', newAuthenticator)$

$\langle 8 \rangle 1.$ $LL1HistoryStateBindingAuthenticated(uniqHistoryStateBinding2)'$
BY $\langle 5 \rangle 3$

$\langle 8 \rangle 2.$ UNCHANGED $LL1NVRAM.symmetricKey$
BY $\langle 2 \rangle 1$, $SymmetricKeyConstantLemma$

$\langle 8 \rangle 3.$ $LL1ObservedAuthenticators' =$
$\quad LL1ObservedAuthenticators \cup \{newAuthenticator\}$
BY $\langle 4 \rangle 1$ DEF $newAuthenticator$, $newHistoryStateBinding$, $newStateHash$, $newHistorySummary$, $newPrivateStateEnc$, $sResult$, $privateState$

$\langle 8 \rangle 4.$ QED
BY $\langle 8 \rangle 1$, $\langle 8 \rangle 2$, $\langle 8 \rangle 3$ DEF $LL1HistoryStateBindingAuthenticated$

We then prove that the history state binding is not authenticated in the unprimed state. We use proof by contradiction. We show that if the history state binding were authenticated in the unprimed state, we could conclude a hash cardinality inequality that contradicts the hash cardinality inequality we proved above.

$\langle 7 \rangle 2.$ $\neg LL1HistoryStateBindingAuthenticated(uniqHistoryStateBinding2')$

$\langle 8 \rangle 1.$ SUFFICES
ASSUME $LL1HistoryStateBindingAuthenticated(uniqHistoryStateBinding2')$
PROVE FALSE
OBVIOUS

$\langle 8 \rangle 2.$ $HashCardinality(LL1NVRAM.historySummary') \leq$
$\quad HashCardinality(LL1NVRAM.historySummary)$

$\langle 9 \rangle 1.$ $CardinalityInvariant$
BY $\langle 2 \rangle 1$

$\langle 9 \rangle 2.$ $LL1NVRAM.historySummary' \in HashType$
BY $\langle 4 \rangle 4$

$\langle 9 \rangle 3.$ $stateHash2 \in HashType$
BY $\langle 5 \rangle 2$

$\langle 9 \rangle 4.$ $LL1NVRAMHistorySummaryUncorrupted$
BY $\langle 4 \rangle 5$

$\langle 9 \rangle 5.$ QED
BY $\langle 8 \rangle 1$, $\langle 9 \rangle 1$, $\langle 9 \rangle 2$, $\langle 9 \rangle 3$, $\langle 9 \rangle 4$ DEF $CardinalityInvariant$, $uniqHistoryStateBinding2$

$\langle 8 \rangle 3.$ QED
BY $\langle 5 \rangle 4$, $\langle 5 \rangle 5$, $\langle 5 \rangle 6$, $\langle 8 \rangle 2$, $GEQorLT$

$\langle 7 \rangle 3.$ QED
BY $\langle 7 \rangle 1$, $\langle 7 \rangle 2$

Since the new authenticator authenticates the new history state binding defined in the *LL1PerformOperation* action, it follows that the history state bindings are equal, by the *MACCollisionResistant* property.

We first have to prove some basic types, then we can appeal to the *MACCollisionResistant* property.

$\langle 6 \rangle 2.$ $LL1NVRAM.symmetricKey \in SymmetricKeyType$
BY $\langle 4 \rangle 3$

$\langle 6 \rangle 3.$ $uniqHistoryStateBinding2' \in HashType$

$\langle 7 \rangle 1.$ $LL1NVRAM.historySummary' \in HashDomain$

$\langle 8 \rangle 1.$ $LL1NVRAM.historySummary' \in HashType$

$\langle 9 \rangle 1.$ $LL1TypeInvariant'$
BY $\langle 2 \rangle 1$

$\langle 9 \rangle 2.$ QED

BY $\langle 9 \rangle 1$, $LL1SubtypeImplicationLemma$ DEF $LL1SubtypeImplication$

$\langle 8 \rangle 2$. QED

BY $\langle 8 \rangle 1$ DEF $HashDomain$

$\langle 7 \rangle 2$. $stateHash2 \in HashDomain$

$\langle 8 \rangle 1$. $stateHash2 \in HashType$

BY $\langle 5 \rangle 2$

$\langle 8 \rangle 2$. QED

BY $\langle 3 \rangle 1$ DEF $HashDomain$

$\langle 7 \rangle 3$. QED

BY $\langle 7 \rangle 1$, $\langle 7 \rangle 2$, $HashTypeSafe$ DEF $uniqHistoryStateBinding2$

$\langle 6 \rangle 4$. $newHistoryStateBinding \in HashType$

BY $\langle 4 \rangle 2$

$\langle 6 \rangle 5$. QED

BY $\langle 6 \rangle 1$, $\langle 6 \rangle 2$, $\langle 6 \rangle 3$, $\langle 6 \rangle 4$, $MACCollisionResistant$ DEF $newAuthenticator$

Since each of the history state bindings is equal to the history state binding defined by the $LL1PerformOperation$ action, the two history state bindings must be equal to each other.

$\langle 5 \rangle 9$. $uniqHistoryStateBinding1' = uniqHistoryStateBinding2'$

BY $\langle 5 \rangle 7$, $\langle 5 \rangle 8$

Since the inputs-state bindings are equal, it follows that the state hashes are equal, by the $HashCollisionResistant$ property.

$\langle 5 \rangle 10$. QED

$\langle 6 \rangle 1$. $LL1NVRAM.historySummary' \in HashDomain$

$\langle 7 \rangle 1$. $LL1NVRAM.historySummary' \in HashType$

BY $\langle 4 \rangle 4$

$\langle 7 \rangle 2$. QED

BY $\langle 7 \rangle 1$ DEF $HashDomain$

$\langle 6 \rangle 2$. $stateHash1 \in HashDomain$

$\langle 7 \rangle 1$. $stateHash1 \in HashType$

BY $\langle 5 \rangle 2$

$\langle 7 \rangle 2$. QED

BY $\langle 7 \rangle 1$ DEF $HashDomain$

$\langle 6 \rangle 3$. $stateHash2 \in HashDomain$

$\langle 7 \rangle 1$. $stateHash2 \in HashType$

BY $\langle 5 \rangle 2$

$\langle 7 \rangle 2$. QED

BY $\langle 7 \rangle 1$ DEF $HashDomain$

$\langle 6 \rangle 4$. QED

BY $\langle 5 \rangle 9$, $\langle 6 \rangle 1$, $\langle 6 \rangle 2$, $\langle 6 \rangle 3$, $HashCollisionResistant$

DEF $uniqHistoryStateBinding1$, $uniqHistoryStateBinding2$

$\langle 4 \rangle 9$. QED

BY $\langle 4 \rangle 6$, $\langle 4 \rangle 7$, $\langle 4 \rangle 8$

The $LL1RepeatOperation$ case is a straightforward application of the $InclusionUnchangedLemma$, the $CardinalityUnchangedLemma$, and the $UniquenessUnchangedLemma$, the preconditions for which follow from the $LL1RepeatOperationUnchangedObseredOutputsLemma$ and the $LL1RepeatOperationUnchangedAuthenticatedHistoryStateBindingsLemma$.

$\langle 3 \rangle 3$. CASE $LL1RepeatOperation$

$\langle 4 \rangle 1$. PICK $input \in LL1AvailableInputs : LL1RepeatOperation!(input)!1$

BY $\langle 3 \rangle 3$ DEF $LL1RepeatOperation$

$\langle 4 \rangle 2$. UNCHANGED $LL1NVRAM$

BY $\langle 4 \rangle 1$

$\langle 4 \rangle 3$. UNCHANGED $LL1ObservedOutputs$

$\langle 5 \rangle 1$. $LL1TypeInvariant \land UnforgeabilityInvariant \land InclusionInvariant$

BY ⟨2⟩1
⟨5⟩2. QED
BY ⟨3⟩3, ⟨5⟩1, *LL1RepeatOperationUnchangedObseredOutputsLemma*
⟨4⟩4. $\forall$ *historyStateBinding* $\in$ *HashType* :
UNCHANGED *LL1HistoryStateBindingAuthenticated*(*historyStateBinding*)
⟨5⟩1. *LL1TypeInvariant* $\wedge$ *UnforgeabilityInvariant* $\wedge$ *InclusionInvariant*
BY ⟨2⟩1
⟨5⟩2. QED
BY ⟨3⟩3, ⟨5⟩1, *LL1RepeatOperationUnchangedAuthenticatedHistoryStateBindingsLemma*
⟨4⟩5. *InclusionInvariant*′
⟨5⟩1. *InclusionInvariant* $\wedge$ *LL1TypeInvariant* $\wedge$ *LL1TypeInvariant*′
BY ⟨2⟩1
⟨5⟩2. QED
BY ⟨4⟩2, ⟨4⟩3, ⟨4⟩4, ⟨5⟩1, *InclusionUnchangedLemma*
⟨4⟩6. *CardinalityInvariant*′
⟨5⟩1. *CardinalityInvariant* $\wedge$ *LL1TypeInvariant*
BY ⟨2⟩1
⟨5⟩2. QED
BY ⟨4⟩2, ⟨4⟩3, ⟨4⟩4, ⟨5⟩1, *CardinalityUnchangedLemma*
⟨4⟩7. *UniquenessInvariant*′
⟨5⟩1. *UniquenessInvariant* $\wedge$ *LL1TypeInvariant*
BY ⟨2⟩1
⟨5⟩2. QED
BY ⟨4⟩2, ⟨4⟩3, ⟨4⟩4, ⟨5⟩1, *UniquenessUnchangedLemma*
⟨4⟩8. QED
BY ⟨4⟩5, ⟨4⟩6, ⟨4⟩7

The *LL1Restart* case is a straightforward application of the *InclusionUnchangedLemma*, the *CardinalityUnchangedLemma*, and the *UniquenessUnchangedLemma*.

⟨3⟩4. CASE *LL1Restart*
⟨4⟩1. UNCHANGED ⟨*LL1NVRAM*, *LL1ObservedOutputs*⟩
BY ⟨3⟩4 DEF *LL1Restart*
⟨4⟩2. $\forall$ *historyStateBinding* $\in$ *HashType* :
UNCHANGED *LL1HistoryStateBindingAuthenticated*(*historyStateBinding*)
⟨5⟩1. UNCHANGED ⟨*LL1NVRAM*, *LL1ObservedAuthenticators*⟩
BY ⟨3⟩4 DEF *LL1Restart*
⟨5⟩2. QED
BY ⟨5⟩1, *UnchangedAuthenticatedHistoryStateBindingsLemma*
⟨4⟩3. *InclusionInvariant*′
⟨5⟩1. *InclusionInvariant* $\wedge$ *LL1TypeInvariant* $\wedge$ *LL1TypeInvariant*′
BY ⟨2⟩1
⟨5⟩2. QED
BY ⟨4⟩1, ⟨4⟩2, ⟨5⟩1, *InclusionUnchangedLemma*
⟨4⟩4. *CardinalityInvariant*′
⟨5⟩1. *CardinalityInvariant* $\wedge$ *LL1TypeInvariant*
BY ⟨2⟩1
⟨5⟩2. QED
BY ⟨4⟩1, ⟨4⟩2, ⟨5⟩1, *CardinalityUnchangedLemma*
⟨4⟩5. *UniquenessInvariant*′
⟨5⟩1. *UniquenessInvariant* $\wedge$ *LL1TypeInvariant*
BY ⟨2⟩1
⟨5⟩2. QED
BY ⟨4⟩1, ⟨4⟩2, ⟨5⟩1, *UniquenessUnchangedLemma*

$\langle 4 \rangle 6$. QED
  BY $\langle 4 \rangle 3$, $\langle 4 \rangle 4$, $\langle 4 \rangle 5$

The *LL1ReadDisk* case is a straightforward application of the *InclusionUnchangedLemma*, the *CardinalityUnchangedLemma*, and the *UniquenessUnchangedLemma*.

$\langle 3 \rangle 5$. CASE *LL1ReadDisk*
  $\langle 4 \rangle 1$. UNCHANGED $\langle LL1NVRAM,\ LL1ObservedOutputs \rangle$
    BY $\langle 3 \rangle 5$ DEF *LL1ReadDisk*
  $\langle 4 \rangle 2$. $\forall$ *historyStateBinding* $\in$ *HashType* :
      UNCHANGED *LL1HistoryStateBindingAuthenticated*(*historyStateBinding*)
    $\langle 5 \rangle 1$. UNCHANGED $\langle LL1NVRAM,\ LL1ObservedAuthenticators \rangle$
      BY $\langle 3 \rangle 5$ DEF *LL1ReadDisk*
    $\langle 5 \rangle 2$. QED
      BY $\langle 5 \rangle 1$, *UnchangedAuthenticatedHistoryStateBindingsLemma*
  $\langle 4 \rangle 3$. *InclusionInvariant'*
    $\langle 5 \rangle 1$. *InclusionInvariant* $\land$ *LL1TypeInvariant* $\land$ *LL1TypeInvariant'*
      BY $\langle 2 \rangle 1$
    $\langle 5 \rangle 2$. QED
      BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$, $\langle 5 \rangle 1$, *InclusionUnchangedLemma*
  $\langle 4 \rangle 4$. *CardinalityInvariant'*
    $\langle 5 \rangle 1$. *CardinalityInvariant* $\land$ *LL1TypeInvariant*
      BY $\langle 2 \rangle 1$
    $\langle 5 \rangle 2$. QED
      BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$, $\langle 5 \rangle 1$, *CardinalityUnchangedLemma*
  $\langle 4 \rangle 5$. *UniquenessInvariant'*
    $\langle 5 \rangle 1$. *UniquenessInvariant* $\land$ *LL1TypeInvariant*
      BY $\langle 2 \rangle 1$
    $\langle 5 \rangle 2$. QED
      BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$, $\langle 5 \rangle 1$, *UniquenessUnchangedLemma*
  $\langle 4 \rangle 6$. QED
    BY $\langle 4 \rangle 3$, $\langle 4 \rangle 4$, $\langle 4 \rangle 5$

The *LL1WriteDisk* case is a straightforward application of the *InclusionUnchangedLemma*, the *CardinalityUnchangedLemma*, and the *UniquenessUnchangedLemma*.

$\langle 3 \rangle 6$. CASE *LL1WriteDisk*
  $\langle 4 \rangle 1$. UNCHANGED $\langle LL1NVRAM,\ LL1ObservedOutputs \rangle$
    BY $\langle 3 \rangle 6$ DEF *LL1WriteDisk*
  $\langle 4 \rangle 2$. $\forall$ *historyStateBinding* $\in$ *HashType* :
      UNCHANGED *LL1HistoryStateBindingAuthenticated*(*historyStateBinding*)
    $\langle 5 \rangle 1$. UNCHANGED $\langle LL1NVRAM,\ LL1ObservedAuthenticators \rangle$
      BY $\langle 3 \rangle 6$ DEF *LL1WriteDisk*
    $\langle 5 \rangle 2$. QED
      BY $\langle 5 \rangle 1$, *UnchangedAuthenticatedHistoryStateBindingsLemma*
  $\langle 4 \rangle 3$. *InclusionInvariant'*
    $\langle 5 \rangle 1$. *InclusionInvariant* $\land$ *LL1TypeInvariant* $\land$ *LL1TypeInvariant'*
      BY $\langle 2 \rangle 1$
    $\langle 5 \rangle 2$. QED
      BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$, $\langle 5 \rangle 1$, *InclusionUnchangedLemma*
  $\langle 4 \rangle 4$. *CardinalityInvariant'*
    $\langle 5 \rangle 1$. *CardinalityInvariant* $\land$ *LL1TypeInvariant*
      BY $\langle 2 \rangle 1$
    $\langle 5 \rangle 2$. QED
      BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$, $\langle 5 \rangle 1$, *CardinalityUnchangedLemma*
  $\langle 4 \rangle 5$. *UniquenessInvariant'*

$\langle 5\rangle 1.$ $\mathit{UniquenessInvariant} \land \mathit{LL1TypeInvariant}$
BY $\langle 2\rangle 1$
$\langle 5\rangle 2.$ QED
BY $\langle 4\rangle 1$, $\langle 4\rangle 2$, $\langle 5\rangle 1$, $\mathit{UniquenessUnchangedLemma}$
$\langle 4\rangle 6.$ QED
BY $\langle 4\rangle 3$, $\langle 4\rangle 4$, $\langle 4\rangle 5$

The $\mathit{LL1CorruptRAM}$ case is a straightforward application of the $\mathit{InclusionUnchangedLemma}$, the $\mathit{CardinalityUnchangedLemma}$, and the $\mathit{UniquenessUnchangedLemma}$.

$\langle 3\rangle 7.$ CASE $\mathit{LL1CorruptRAM}$
$\langle 4\rangle 1.$ UNCHANGED $\langle \mathit{LL1NVRAM}, \mathit{LL1ObservedOutputs} \rangle$
BY $\langle 3\rangle 7$ DEF $\mathit{LL1CorruptRAM}$
$\langle 4\rangle 2.$ $\forall\, \mathit{historyStateBinding} \in \mathit{HashType}:$
UNCHANGED $\mathit{LL1HistoryStateBindingAuthenticated}(\mathit{historyStateBinding})$
$\langle 5\rangle 1.$ UNCHANGED $\langle \mathit{LL1NVRAM}, \mathit{LL1ObservedAuthenticators} \rangle$
BY $\langle 3\rangle 7$ DEF $\mathit{LL1CorruptRAM}$
$\langle 5\rangle 2.$ QED
BY $\langle 5\rangle 1$, $\mathit{UnchangedAuthenticatedHistoryStateBindingsLemma}$
$\langle 4\rangle 3.$ $\mathit{InclusionInvariant}'$
$\langle 5\rangle 1.$ $\mathit{InclusionInvariant} \land \mathit{LL1TypeInvariant} \land \mathit{LL1TypeInvariant}'$
BY $\langle 2\rangle 1$
$\langle 5\rangle 2.$ QED
BY $\langle 4\rangle 1$, $\langle 4\rangle 2$, $\langle 5\rangle 1$, $\mathit{InclusionUnchangedLemma}$
$\langle 4\rangle 4.$ $\mathit{CardinalityInvariant}'$
$\langle 5\rangle 1.$ $\mathit{CardinalityInvariant} \land \mathit{LL1TypeInvariant}$
BY $\langle 2\rangle 1$
$\langle 5\rangle 2.$ QED
BY $\langle 4\rangle 1$, $\langle 4\rangle 2$, $\langle 5\rangle 1$, $\mathit{CardinalityUnchangedLemma}$
$\langle 4\rangle 5.$ $\mathit{UniquenessInvariant}'$
$\langle 5\rangle 1.$ $\mathit{UniquenessInvariant} \land \mathit{LL1TypeInvariant}$
BY $\langle 2\rangle 1$
$\langle 5\rangle 2.$ QED
BY $\langle 4\rangle 1$, $\langle 4\rangle 2$, $\langle 5\rangle 1$, $\mathit{UniquenessUnchangedLemma}$
$\langle 4\rangle 6.$ QED
BY $\langle 4\rangle 3$, $\langle 4\rangle 4$, $\langle 4\rangle 5$

The $\mathit{LL1RestrictedCorruption}$ case is non-trivial, although nowhere near as involved as the $\mathit{LL1PerformOperation}$ case.

$\langle 3\rangle 8.$ CASE $\mathit{LL1RestrictedCorruption}$

We pick a garbage history summary in the appropriate type.

$\langle 4\rangle 1.$ PICK $\mathit{garbageHistorySummary} \in \mathit{HashType}:$
$\mathit{LL1RestrictedCorruption}!\mathit{nvram}!(\mathit{garbageHistorySummary})$
BY $\langle 3\rangle 8$ DEF $\mathit{LL1RestrictedCorruption}$

The primed value of the history summary in the NVRAM equals this garbage history summary.

$\langle 4\rangle 2.$ $\mathit{LL1NVRAM}.\mathit{historySummary}' = \mathit{garbageHistorySummary}$
$\langle 5\rangle 1.$ $\mathit{LL1NVRAM}' = [\mathit{historySummary} \mapsto \mathit{garbageHistorySummary},$
$\mathit{symmetricKey} \mapsto \mathit{LL1NVRAM}.\mathit{symmetricKey}]$
BY $\langle 4\rangle 1$
$\langle 5\rangle 2.$ QED
BY $\langle 5\rangle 1$

We now prove each invariant separately, starting with the $\mathit{InclusionInvariant}$.

$\langle 4\rangle 3.$ $\mathit{InclusionInvariant}'$

To prove the universally quantified expression, we take a new set of variables of the appropriate types. For the TAKE step to be meaningful to the prover, first we have to tell the prover to expand the definition of *InclusionInvariant*, so it will see the universally quantified expression therein.

$\langle 5 \rangle$ USE DEF $InclusionInvariant$
$\langle 5 \rangle 1.$ TAKE $input \in InputType,$
$\quad historySummary \in HashType,$
$\quad publicState \in PublicStateType,$
$\quad privateStateEnc \in PrivateStateEncType$

To simplify the writing of the proof, we re-state the definitions from the *InclusionInvariant*.

$\langle 5 \rangle$ $inclStateHash \triangleq Hash(publicState, privateStateEnc)$
$\langle 5 \rangle$ $inclHistoryStateBinding \triangleq Hash(historySummary, inclStateHash)$
$\langle 5 \rangle$ $inclPrivateState \triangleq SymmetricDecrypt(LL1NVRAM.symmetricKey, privateStateEnc)$
$\langle 5 \rangle$ $inclSResult \triangleq Service(publicState, inclPrivateState, input)$
$\langle 5 \rangle$ $inclNewPrivateStateEnc \triangleq$
$\quad SymmetricEncrypt(LL1NVRAM.symmetricKey, inclSResult.newPrivateState)$
$\langle 5 \rangle$ $inclNewStateHash \triangleq Hash(inclSResult.newPublicState, inclNewPrivateStateEnc)$
$\langle 5 \rangle$ $inclNewHistoryStateBinding \triangleq Hash(LL1NVRAM.historySummary, inclNewStateHash)$

We then assert the type safety of these definitions, with the help of the *InclusionInvariantDefsTypeSafeLemma*.

$\langle 5 \rangle 2.$ $\land\ inclStateHash \in HashType$
$\quad \land\ inclHistoryStateBinding \in HashType$
$\quad \land\ inclPrivateState \in PrivateStateType$
$\quad \land\ inclSResult \in ServiceResultType$
$\quad \land\ inclSResult.newPublicState \in PublicStateType$
$\quad \land\ inclSResult.newPrivateState \in PrivateStateType$
$\quad \land\ inclSResult.output \in OutputType$
$\quad \land\ inclNewPrivateStateEnc \in PrivateStateEncType$
$\quad \land\ inclNewStateHash \in HashType$
$\quad \land\ inclNewHistoryStateBinding \in HashType$
$\langle 6 \rangle 1.$ $LL1TypeInvariant$
BY $\langle 2 \rangle 1$
$\langle 6 \rangle 2.$ QED
BY $\langle 5 \rangle 1$, $\langle 6 \rangle 1$, $InclusionInvariantDefsTypeSafeLemma$

The *InclusionInvariant* states an implication. To prove this, it suffices to prove that the the antecedent is false. The antecedent is a conjunction. We will prove that the first conjunct implies that the second conjunct is false.

$\langle 5 \rangle 3.$ SUFFICES
ASSUME TRUE
PROVE
$\quad LL1NVRAM.historySummary' = Hash(historySummary, input) \Rightarrow$
$\quad\quad \neg LL1HistoryStateBindingAuthenticated(inclHistoryStateBinding)'$
OBVIOUS

We hide the definition of *InclusionInvariant* and the definitions from the *InclusionInvariant*.

$\langle 5 \rangle$ HIDE DEF $InclusionInvariant$
$\langle 5 \rangle$ HIDE DEF $inclStateHash, inclHistoryStateBinding, inclPrivateState, inclSResult,$
$\quad inclNewPrivateStateEnc, inclNewStateHash, inclNewHistoryStateBinding$

We assume the antecedent in this conjunction.

$\langle 5 \rangle 4.$ HAVE $LL1NVRAM.historySummary' = Hash(historySummary, input)$

We prove that all of the authenticators in the set of observed authenticators fail to validate the history state binding defined by the inclusion invariant in the unprimed state.

$\langle 5 \rangle 5.$ $\forall\, authenticator \in LL1ObservedAuthenticators :$
$\quad \neg\, ValidateMAC(LL1NVRAM.symmetricKey, inclHistoryStateBinding, authenticator)$

We will make use of the conjunct in *LL1RestrictedCorruption* that prevents the garbage history summary from being a predecessor to any history summary in an authenticated history state binding. This conjunct states a 4-way universally quantifiied predicate.

$\langle 6\rangle 1.$ $LL1RestrictedCorruption!nvram!previous(garbageHistorySummary)$
BY $\langle 4\rangle 1$

We will pare the predicate down to a single universal quantification by showing particular instances for three of the quantifiers.

$\langle 6\rangle 2.$ $inclStateHash \in HashType$
BY $\langle 5\rangle 2$

$\langle 6\rangle 3.$ $historySummary \in HashType$
BY $\langle 5\rangle 1$

$\langle 6\rangle 4.$ $input \in InputType$
BY $\langle 5\rangle 1$

We also need to show that the antecedent of the implication in this predicate is satisfied.

$\langle 6\rangle 5.$ $garbageHistorySummary = Hash(historySummary, input)$
BY $\langle 4\rangle 2$, $\langle 5\rangle 4$

The consequent of the implication in this predicate is exactly what we need for our conclusion.

$\langle 6\rangle 6.$ QED
BY $\langle 6\rangle 1$, $\langle 6\rangle 2$, $\langle 6\rangle 3$, $\langle 6\rangle 4$, $\langle 6\rangle 5$ DEF *inclHistoryStateBinding*

We show that the symmetric key in the *NVRAM* has not changed, nor has the set of observed authenticators.

$\langle 5\rangle 6.$ UNCHANGED $LL1NVRAM.symmetricKey$
BY $\langle 2\rangle 1$, *SymmetricKeyConstantLemma*

$\langle 5\rangle 7.$ UNCHANGED $LL1ObservedAuthenticators$
BY $\langle 3\rangle 8$ DEF *LL1RestrictedCorruption*

Thus, we can conclude that all of the authenticators in the set of observed authenticators fail to validate the history state binding defined by the inclusion invariant in the primed state. This satisfies the definition of *LL1HistoryStateBindingAuthenticated*.

$\langle 5\rangle 8.$ QED
BY $\langle 5\rangle 5$, $\langle 5\rangle 6$, $\langle 5\rangle 7$ DEF *LL1HistoryStateBindingAuthenticated*

We next prove the *CardinalityInvariant*.

$\langle 4\rangle 4.$ $CardinalityInvariant'$

To prove the universally quantified expression, we take a new set of variables of the appropriate types. For the TAKE step to be meaningful to the prover, first we have to tell the prover to expand the definition of *CardinalityInvariant*, so it will see the universally quantified expression therein.

$\langle 5\rangle$ USE DEF *CardinalityInvariant*

$\langle 5\rangle 1.$ TAKE $historySummary \in HashType$, $stateHash \in HashType$

The *CardinalityInvariant* states an implication. To prove this, it suffices to prove that the the antecedent is false. The antecedent is a conjunction. We will prove that the first conjunct is false.

$\langle 5\rangle 2.$ SUFFICES ASSUME TRUEPROVE $\neg LL1NVRAMHistorySummaryUncorrupted'$
OBVIOUS

We then hide the definition.

$\langle 5\rangle$ HIDE DEF *CardinalityInvariant*

We will make use of the conjunct in *LL1RestrictedCorruption* that prevents the garbage history summary from being in an authenticated history state binding.

$\langle 5\rangle 3.$ $LL1RestrictedCorruption!nvram!current(garbageHistorySummary)$
BY $\langle 4\rangle 1$

The following equivalence, plus the knowledge that the symmetric key in the *NVRAM* and the set of observed authenticators have not changed, are sufficient to prove the conclusion.

$\langle 5\rangle 4.$ $LL1NVRAM.historySummary' = garbageHistorySummary$
BY $\langle 4\rangle 2$

⟨5⟩5. UNCHANGED $LL1NVRAM.symmetricKey$
BY ⟨2⟩1, *SymmetricKeyConstantLemma*
⟨5⟩6. UNCHANGED *LL1ObservedAuthenticators*
BY ⟨3⟩8 DEF *LL1RestrictedCorruption*
⟨5⟩7. QED
BY ⟨5⟩3, ⟨5⟩4, ⟨5⟩5, ⟨5⟩6
DEF *LL1NVRAMHistorySummaryUncorrupted*, *LL1HistoryStateBindingAuthenticated*

We last prove the *UniquenessInvariant*.

⟨4⟩5. $UniquenessInvariant'$

To prove the universally quantified expression, we take a new set of variables of the appropriate types. For the TAKE step to be meaningful to the prover, first we have to tell the prover to expand the definition of *UniquenessInvariant*, so it will see the universally quantified expression therein.

⟨5⟩ USE DEF *UniquenessInvariant*
⟨5⟩1. TAKE $stateHash1, stateHash2 \in HashType$

To simplify the writing of the proof, we re-state the definitions from the *UniquenessInvariant*.

⟨5⟩ $uniqHistoryStateBinding1 \triangleq Hash(LL1NVRAM.historySummary, stateHash1)$
⟨5⟩ $uniqHistoryStateBinding2 \triangleq Hash(LL1NVRAM.historySummary, stateHash2)$

The *UniquenessInvariant* states an implication. To prove this, it suffices to prove that the the antecedent is false. The antecedent is a conjunction. We will prove that the first conjunct is false.

⟨5⟩2. SUFFICES
ASSUME TRUE
PROVE $\neg LL1HistoryStateBindingAuthenticated(uniqHistoryStateBinding1)'$
OBVIOUS

We hide the definitions of *UniquenessInvariant* and the definitions from the *UniquenessInvariant*.

⟨5⟩ HIDE DEF *UniquenessInvariant*
⟨5⟩ HIDE DEF *uniqHistoryStateBinding*1, *uniqHistoryStateBinding*2

We will make use of the conjunct in *LL1RestrictedCorruption* that prevents the garbage history summary from being in an authenticated history state binding. This conjunct states a 2-way universally quantifiied predicate.

⟨5⟩3. $LL1RestrictedCorruption!nvram!current(garbageHistorySummary)$
BY ⟨4⟩1

We will pare the predicate down to a single universal quantification by showing particular instances for one of the quantifiers.

⟨5⟩4. $stateHash1 \in HashType$
BY ⟨5⟩1

The following equivalence, plus the knowledge that the symmetric key in the *NVRAM* and the set of observed authenticators have not changed, are sufficient to prove the conclusion.

⟨5⟩5. $LL1NVRAM.historySummary' = garbageHistorySummary$
BY ⟨4⟩2
⟨5⟩6. UNCHANGED $LL1NVRAM.symmetricKey$
BY ⟨2⟩1, *SymmetricKeyConstantLemma*
⟨5⟩7. UNCHANGED *LL1ObservedAuthenticators*
BY ⟨3⟩8 DEF *LL1RestrictedCorruption*
⟨5⟩8. QED
BY ⟨5⟩3, ⟨5⟩4, ⟨5⟩5, ⟨5⟩6, ⟨5⟩7
DEF *uniqHistoryStateBinding*1, *LL1HistoryStateBindingAuthenticated*
⟨4⟩6. QED
BY ⟨4⟩3, ⟨4⟩4, ⟨4⟩5
⟨3⟩9. QED
BY ⟨2⟩3, ⟨3⟩1, ⟨3⟩2, ⟨3⟩3, ⟨3⟩4, ⟨3⟩5, ⟨3⟩6, ⟨3⟩7, ⟨3⟩8 DEF *LL1Next*
⟨2⟩4. QED
BY ⟨2⟩1, ⟨2⟩2, ⟨2⟩3

$\langle 1 \rangle 5.$ QED

Using the *Inv*1 proof rule, the base case and the induction step together imply that the invariant always holds.

$\langle 2 \rangle 1.$ $(\wedge\ \mathit{InclusionInvariant}$
$\wedge\ \mathit{CardinalityInvariant}$
$\wedge\ \mathit{UniquenessInvariant}$
$\wedge\ \Box[\mathit{LL1Next}]_{\mathit{LL1Vars}}$
$\wedge\ \Box \mathit{LL1TypeInvariant})$
$\Rightarrow$
$\Box \mathit{InclusionInvariant} \wedge \Box \mathit{CardinalityInvariant} \wedge \Box \mathit{UniquenessInvariant}$

BY $\langle 1 \rangle 4$, *Inv*1

$\langle 2 \rangle 2.$ QED

BY $\langle 1 \rangle 1$, $\langle 1 \rangle 2$, $\langle 1 \rangle 3$, $\langle 2 \rangle 1$ DEF *LL1Spec*

From the above proof and the previous proof that the *UnforgeabilityInvariant* is an inductive invariant of the Memoir-Basic spec, it proves that the set of *CorrectnessInvariants* are inductive invariants of the Memoir-Basic spec.

THEOREM $\mathit{CorrectnessInvariance} \triangleq \mathit{LL1Spec} \Rightarrow \Box \mathit{CorrectnessInvariants}$

$\langle 1 \rangle 1.$ $\mathit{LL1Spec} \Rightarrow \Box \mathit{UnforgeabilityInvariant}$

BY *UnforgeabilityInvariance*

$\langle 1 \rangle 2.$ $\mathit{LL1Spec} \Rightarrow \Box \mathit{InclusionInvariant} \wedge \Box \mathit{UniquenessInvariant}$

BY *InclusionCardinalityUniquenessInvariance*

$\langle 1 \rangle 3.$ $\mathit{UnforgeabilityInvariant} \wedge \mathit{InclusionInvariant} \wedge \mathit{UniquenessInvariant} \Rightarrow \mathit{CorrectnessInvariants}$

BY DEF *CorrectnessInvariants*

$\langle 1 \rangle 4.$ QED

BY $\langle 1 \rangle 1$, $\langle 1 \rangle 2$, $\langle 1 \rangle 3$

4.7 Proof that Memoir-Basic Spec Implements High-Level Spec

MODULE *MemoirLL1Implementation*

This module proves that the Memoir-Basic spec implements the high-level spec, under the defined refinement. It begins with two supporting lemmas:
NonAdvancementLemma
LL1NVRAMHistorySummaryUncorruptedEqualsHLAliveLemma
Then, it states and proves the *LL1Implementation* theorem.

EXTENDS *MemoirLL1InclCardUniqInvariance*

The *NonAdvancementLemma* proves that, if there is no change to the *NVRAM* or to the authentication status of any history state binding, then the high-level public and private state defined by the refinement both stutter.

THEOREM $NonAdvancementLemma \triangleq$

$(\land\ LL1Refinement$
$\land\ LL1Refinement'$
$\land\ LL1TypeInvariant$
$\land\ LL1TypeInvariant'$
$\land\ UniquenessInvariant$
$\land\ \text{UNCHANGED } LL1NVRAM$
$\land\ \forall\, historyStateBinding \in HashType :$
$\qquad \text{UNCHANGED } LL1HistoryStateBindingAuthenticated(historyStateBinding))$
$\Rightarrow$
$\quad \text{UNCHANGED } \langle HLPublicState,\ HLPrivateState \rangle$

We assume the antecedent.

$\langle 1 \rangle 1.$ HAVE $\land\ LL1Refinement$
$\land\ LL1Refinement'$
$\land\ LL1TypeInvariant$
$\land\ LL1TypeInvariant'$
$\land\ UniquenessInvariant$
$\land\ \text{UNCHANGED } LL1NVRAM$
$\land\ \forall\, historyStateBinding \in HashType :$
$\qquad \text{UNCHANGED } LL1HistoryStateBindingAuthenticated(historyStateBinding)$

$\langle 1 \rangle 2.$ $LL1TypeInvariant \land LL1TypeInvariant'$
BY $\langle 1 \rangle 1$

These definitions are copied from the *LL1Refinement*.

$\langle 1 \rangle$ $refPrivateStateEnc \triangleq SymmetricEncrypt(LL1NVRAM.symmetricKey,\ HLPrivateState)$
$\langle 1 \rangle$ $refStateHash \triangleq Hash(HLPublicState,\ refPrivateStateEnc)$
$\langle 1 \rangle$ $refHistoryStateBinding \triangleq Hash(LL1NVRAM.historySummary,\ refStateHash)$

We prove that the definitions satisfy their types in both the unprimed and primed states, using the *LL1RefinementDefsTypeSafeLemma* and the *LL1RefinementPrimeDefsTypeSafeLemma*.

$\langle 1 \rangle 3.$ $\land\ refPrivateStateEnc \in PrivateStateEncType$
$\land\ refStateHash \in HashType$
$\land\ refHistoryStateBinding \in HashType$
BY $\langle 1 \rangle 1$, $\langle 1 \rangle 2$, $LL1RefinementDefsTypeSafeLemma$

$\langle 1 \rangle 4.$ $\land\ refPrivateStateEnc' \in PrivateStateEncType$
$\land\ refStateHash' \in HashType$
$\land\ refHistoryStateBinding' \in HashType$
BY $\langle 1 \rangle 1$, $\langle 1 \rangle 2$, $LL1RefinementPrimeDefsTypeSafeLemma$

We then hide the definitions. We'll pull them in as needed later.

$\langle 1 \rangle$ HIDE DEF $refPrivateStateEnc,\ refStateHash,\ refHistoryStateBinding$

We consider the two possible states of the *LL1NVRAMHistorySummaryUncorrupted* predicate separately. In the first case, that the history state binding in the *NVRAM* is authenticated in the unprimed state.

$\langle 1 \rangle 5.$ CASE *LL1NVRAMHistorySummaryUncorrupted*

We first prove that the history state binding in the *NVRAM* is authenticated in the primed state, as well. This will be needed a few places below.

$\langle 2 \rangle 1.$ *LL1NVRAMHistorySummaryUncorrupted*$'$

$\langle 3 \rangle 1.$ *LL1NVRAMHistorySummaryUncorrupted*

BY $\langle 1 \rangle 5$

$\langle 3 \rangle 2.$ UNCHANGED *LL1NVRAMHistorySummaryUncorrupted*

$\langle 4 \rangle 1.$ $\land$ *LL1TypeInvariant*

$\land$ UNCHANGED *LL1NVRAM*

$\land$ $\forall$ *historyStateBinding* $\in$ *HashType* :

UNCHANGED *LL1HistoryStateBindingAuthenticated*(*historyStateBinding*)

BY $\langle 1 \rangle 1$

$\langle 4 \rangle 2.$ QED

BY $\langle 4 \rangle 1$, *LL1NVRAMHistorySummaryUncorruptedUnchangedLemma*

$\langle 3 \rangle 3.$ QED

BY $\langle 3 \rangle 1$, $\langle 3 \rangle 2$

We show that the refined public state does not change and that the refined encrypted private state does not change. We will use the *HashCollisionResistant* property.

$\langle 2 \rangle 2.$ $\land$ UNCHANGED *HLPublicState*

$\land$ UNCHANGED *refPrivateStateEnc*

We begin by proving some types we will need to appeal to the *HashCollisionResistant* property.

$\langle 3 \rangle 1.$ *HLPublicState* $\in$ *HashDomain*

$\langle 4 \rangle 1.$ *HLPublicState* $\in$ *PublicStateType*

$\langle 5 \rangle 1.$ *LL1Refinement*

BY $\langle 1 \rangle 1$

$\langle 5 \rangle 2.$ QED

BY $\langle 1 \rangle 5$, $\langle 5 \rangle 1$ DEF *LL1Refinement*

$\langle 4 \rangle 2.$ QED

BY $\langle 4 \rangle 1$ DEF *HashDomain*

$\langle 3 \rangle 2.$ *HLPublicState*$'$ $\in$ *HashDomain*

$\langle 4 \rangle 1.$ *HLPublicState*$'$ $\in$ *PublicStateType*

$\langle 5 \rangle 1.$ *LL1Refinement*$'$

BY $\langle 1 \rangle 1$

$\langle 5 \rangle 2.$ QED

BY $\langle 2 \rangle 1$, $\langle 5 \rangle 1$ DEF *LL1Refinement*

$\langle 4 \rangle 2.$ QED

BY $\langle 4 \rangle 1$ DEF *HashDomain*

$\langle 3 \rangle 3.$ *refPrivateStateEnc* $\in$ *HashDomain*

$\langle 4 \rangle 1.$ *refPrivateStateEnc* $\in$ *PrivateStateEncType*

BY $\langle 1 \rangle 3$

$\langle 4 \rangle 2.$ QED

BY $\langle 4 \rangle 1$ DEF *HashDomain*

$\langle 3 \rangle 4.$ *refPrivateStateEnc*$'$ $\in$ *HashDomain*

$\langle 4 \rangle 1.$ *refPrivateStateEnc*$'$ $\in$ *PrivateStateEncType*

BY $\langle 1 \rangle 4$

$\langle 4 \rangle 2.$ QED

BY $\langle 4 \rangle 1$ DEF *HashDomain*

We then prove that the refined state hash does not change. We will use the *UniquenessInvariant*.

$\langle 3 \rangle 5.$ UNCHANGED *refStateHash*

We prove some types we will need for the *UniquenessInvariant*.

$\langle 4 \rangle 1.$ $refStateHash \in HashType$
  BY $\langle 1 \rangle 3$

$\langle 4 \rangle 2.$ $refStateHash' \in HashType$
  BY $\langle 1 \rangle 4$

We prove that the unprimed history state binding is authenticated in the unprimed state. This follows directly from the *LL1Refinement*.

$\langle 4 \rangle 3.$ $LL1HistoryStateBindingAuthenticated(refHistoryStateBinding)$
 $\langle 5 \rangle 1.$ $LL1Refinement$
  BY $\langle 1 \rangle 1$
 $\langle 5 \rangle 2.$ $LL1NVRAMHistorySummaryUncorrupted$
  BY $\langle 1 \rangle 5$
 $\langle 5 \rangle 3.$ QED
  BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$ DEF $LL1Refinement$, $refHistoryStateBinding$, $refStateHash$, $refPrivateStateEnc$

We prove that the primed history state binding is authenticated in the unprimed state. This is a little complicated, because TLA+ does not allow us to prime an operator, only an expression. So, we have to expand the definition of *LL1HistoryStateBindingAuthenticated* for the four steps of this proof.

$\langle 4 \rangle 4.$ $LL1HistoryStateBindingAuthenticated(refHistoryStateBinding')$

Step 1: The primed history state binding is authenticated in the primed state. This follows directly from the primed *LL1Refinement*.

 $\langle 5 \rangle 1.$ $\exists\, authenticator \in LL1ObservedAuthenticators' :$
  $ValidateMAC(LL1NVRAM.symmetricKey', refHistoryStateBinding', authenticator)$
  $\langle 6 \rangle 1.$ $LL1HistoryStateBindingAuthenticated(refHistoryStateBinding)'$
   $\langle 7 \rangle 1.$ $LL1Refinement'$
    BY $\langle 1 \rangle 1$
   $\langle 7 \rangle 2.$ $LL1NVRAMHistorySummaryUncorrupted'$
    BY $\langle 2 \rangle 1$
   $\langle 7 \rangle 3.$ QED
    BY $\langle 7 \rangle 1$, $\langle 7 \rangle 2$ DEF $LL1Refinement$, $refHistoryStateBinding$, $refStateHash$, $refPrivateStateEnc$
  $\langle 6 \rangle 2.$ QED
   BY $\langle 6 \rangle 1$ DEF $LL1HistoryStateBindingAuthenticated$

Step 2: The primed history state binding is of *HashType*. We need this so we can apply the assumption that there is no change to the authentication status of any history state binding.

 $\langle 5 \rangle 2.$ $refHistoryStateBinding' \in HashType$
  BY $\langle 1 \rangle 4$

Step 3: There is no change to the authentication status of any history state binding.

 $\langle 5 \rangle 3.$ $\forall\, historyStateBinding \in HashType :$

$$
\begin{array}{l}
(\exists\, authenticator \in LL1ObservedAuthenticators' : \\
\quad ValidateMAC(LL1NVRAM.symmetricKey', historyStateBinding, authenticator)) \\
\Rightarrow \\
(\exists\, authenticator \in LL1ObservedAuthenticators : \\
\quad ValidateMAC(LL1NVRAM.symmetricKey, historyStateBinding, authenticator))
\end{array}
$$

  $\langle 6 \rangle 1.$ $\forall\, historyStateBinding \in HashType :$
   UNCHANGED $LL1HistoryStateBindingAuthenticated(historyStateBinding)$
   BY $\langle 1 \rangle 1$
  $\langle 6 \rangle 2.$ QED
   BY $\langle 6 \rangle 1$ DEF $LL1HistoryStateBindingAuthenticated$

Step 4: The primed history state binding is authenticated in the unprimed state.

 $\langle 5 \rangle 4.$ $\exists\, authenticator \in LL1ObservedAuthenticators :$
  $ValidateMAC(LL1NVRAM.symmetricKey, refHistoryStateBinding', authenticator)$
  BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$, $\langle 5 \rangle 3$

$\langle 5 \rangle 5$. QED

BY $\langle 5 \rangle 4$ DEF *LL1HistoryStateBindingAuthenticated*

Since the authentication status of the history state binding has not changed, the state hash has not changed.

$\langle 4 \rangle 5$. QED

$\langle 5 \rangle 1$. *UniquenessInvariant*

BY $\langle 1 \rangle 1$

$\langle 5 \rangle 2$. UNCHANGED *LL1NVRAM.historySummary*

$\langle 6 \rangle 1$. UNCHANGED *LL1NVRAM*

BY $\langle 1 \rangle 1$

$\langle 6 \rangle 2$. QED

BY $\langle 6 \rangle 1$

$\langle 5 \rangle 3$. QED

BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$, $\langle 4 \rangle 3$, $\langle 4 \rangle 4$, $\langle 5 \rangle 1$, $\langle 5 \rangle 2$

DEF *UniquenessInvariant*, *refHistoryStateBinding*

Since the refined state hash does not change, the refined public state and encrypted private state do not change, because the hash is collision-resistant.

$\langle 3 \rangle 6$. QED

BY $\langle 3 \rangle 1$, $\langle 3 \rangle 2$, $\langle 3 \rangle 3$, $\langle 3 \rangle 4$, $\langle 3 \rangle 5$, *HashCollisionResistant*DEF *refStateHash*

From the previous step, it immediately follows that the refined public state does not change.

$\langle 2 \rangle 3$. UNCHANGED *HLPublicState*

BY $\langle 2 \rangle 2$

It is slightly more involved to show that the refined private state does not change. We need to show that the symmetric key in the *NVRAM* does not change, and we also have to employ the correctness of the symmetric crypto operations.

$\langle 2 \rangle 4$. UNCHANGED *HLPrivateState*

In the unprimed state, the refined private state equals the decryption of the refined encrypted private state, using the unprimed symmetry key in the *NVRAM*. This follows because the refined encrypted private state is defined in the *LL1Refinement* as the encryption of the refined private state, and the symmetric crypto operations are assumed to be correct.

$\langle 3 \rangle 1$. $HLPrivateState = SymmetricDecrypt(LL1NVRAM.symmetricKey, refPrivateStateEnc)$

$\langle 4 \rangle 1$. $LL1NVRAM.symmetricKey \in SymmetricKeyType$

BY $\langle 1 \rangle 2$, *LL1SubtypeImplicationLemma*DEF *LL1SubtypeImplication*

$\langle 4 \rangle 2$. $HLPrivateState \in PrivateStateType$

$\langle 5 \rangle 1$. *LL1Refinement*

BY $\langle 1 \rangle 1$

$\langle 5 \rangle 2$. QED

BY $\langle 1 \rangle 5$, $\langle 5 \rangle 1$ DEF *LL1Refinement*

$\langle 4 \rangle 3$. QED

BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$, *SymmetricCryptoCorrect*DEF *refPrivateStateEnc*

We then show that in the primed state, the refined private state equals the decryption of the refined encrypted private state, using the unprimed symmetry key in the *NVRAM*.

$\langle 3 \rangle 2$. $HLPrivateState' = SymmetricDecrypt(LL1NVRAM.symmetricKey, refPrivateStateEnc)$

In the primed state, the refined private state equals the decryption of the refined encrypted private state, using the primed symmetry key in the *NVRAM*. This follows because the refined encrypted private state is defined in the *LL1Refinement* as the encryption of the refined private state, and the symmetric crypto operations are assumed to be correct.

$\langle 4 \rangle 1$. $HLPrivateState' = SymmetricDecrypt(LL1NVRAM.symmetricKey', refPrivateStateEnc')$

$\langle 5 \rangle 1$. $LL1NVRAM.symmetricKey' \in SymmetricKeyType$

BY $\langle 1 \rangle 2$, *LL1SubtypeImplicationLemma*DEF *LL1SubtypeImplication*

$\langle 5 \rangle 2$. $HLPrivateState' \in PrivateStateType$

$\langle 6 \rangle 1$. $LL1Refinement'$

BY $\langle 1 \rangle 1$

$\langle 6\rangle 2.$ QED

BY $\langle 2\rangle 1$, $\langle 6\rangle 1$ DEF *LL1Refinement*

$\langle 5\rangle 3.$ QED

BY $\langle 5\rangle 1$, $\langle 5\rangle 2$, *SymmetricCryptoCorrect*DEF *refPrivateStateEnc*

The symmetry key in the *NVRAM* is unchanged, and the refined encrypted private state is unchanged.

$\langle 4\rangle 2.$ UNCHANGED *LL1NVRAM.symmetricKey*

$\langle 5\rangle 1.$ UNCHANGED *LL1NVRAM*

BY $\langle 1\rangle 1$

$\langle 5\rangle 2.$ QED

BY $\langle 5\rangle 1$

$\langle 4\rangle 3.$ UNCHANGED *refPrivateStateEnc*

BY $\langle 2\rangle 2$

Since the relevant variables are unchanged, we can conclude that, in the primed state, the refined private state equals the decryption of the refined encrypted private state, using the unprimed symmetry key in the *NVRAM*.

$\langle 4\rangle 4.$ QED

BY $\langle 4\rangle 1$, $\langle 4\rangle 2$, $\langle 4\rangle 3$

Since both the unprimed and primed refined private state are equal to the same expression, they are equal to each other.

$\langle 3\rangle 3.$ QED

BY $\langle 3\rangle 1$, $\langle 3\rangle 2$

$\langle 2\rangle 5.$ QED

BY $\langle 1\rangle 5$, $\langle 2\rangle 3$, $\langle 2\rangle 4$

In the second case, the history state binding in the *NVRAM* is not authenticated in the unprimed state.

$\langle 1\rangle 6.$ CASE $\neg$*LL1NVRAMHistorySummaryUncorrupted*

By the *LL1Refinement*, the high-level public and private states are equal to their dead states.

$\langle 2\rangle 1.$ $\wedge$ *HLPublicState* = *DeadPublicState*
$\wedge$ *HLPrivateState* = *DeadPrivateState*

$\langle 3\rangle 1.$ *LL1Refinement*

BY $\langle 1\rangle 1$

$\langle 3\rangle 2.$ QED

BY $\langle 1\rangle 6$, $\langle 3\rangle 1$ DEF *LL1Refinement*

The history state binding in the *NVRAM* is also not authenticated in the primed state, so again by the *LL1Refinement*, the high-level public and private states are equal to their dead states.

$\langle 2\rangle 2.$ $\wedge$ *HLPublicState*$'$ = *DeadPublicState*
$\wedge$ *HLPrivateState*$'$ = *DeadPrivateState*

$\langle 3\rangle 1.$ $\neg$*LL1NVRAMHistorySummaryUncorrupted*$'$

$\langle 4\rangle 1.$ UNCHANGED *LL1NVRAMHistorySummaryUncorrupted*

$\langle 5\rangle 1.$ $\wedge$ *LL1TypeInvariant*
$\wedge$ UNCHANGED *LL1NVRAM*
$\wedge$ $\forall$ *historyStateBinding* $\in$ *HashType* :
UNCHANGED *LL1HistoryStateBindingAuthenticated*(*historyStateBinding*)

BY $\langle 1\rangle 1$

$\langle 5\rangle 2.$ QED

BY $\langle 5\rangle 1$, *LL1NVRAMHistorySummaryUncorruptedUnchangedLemma*

$\langle 4\rangle 2.$ QED

BY $\langle 1\rangle 6$, $\langle 4\rangle 1$

$\langle 3\rangle 2.$ *LL1Refinement*$'$

BY $\langle 1\rangle 1$

$\langle 3\rangle 3.$ QED

BY $\langle 3\rangle 1$, $\langle 3\rangle 2$ DEF *LL1Refinement*

$\langle 2\rangle 3.$ QED

BY $\langle 2\rangle 1$, $\langle 2\rangle 2$

$\langle 1\rangle 7$. QED

BY $\langle 1\rangle 5$, $\langle 1\rangle 6$

This simple lemma proves that the two predicates *LL1NVRAMHistorySummaryUncorrupted* and *HLAlive* are equal whenever *LL1Refinement* is true, in either unprimed or primed state.

THEOREM $LL1NVRAMHistorySummaryUncorruptedEqualsHLAliveLemma \triangleq$

$\wedge\ LL1Refinement \Rightarrow LL1NVRAMHistorySummaryUncorrupted = HLAlive$

$\wedge\ LL1Refinement' \Rightarrow LL1NVRAMHistorySummaryUncorrupted' = HLAlive'$

$\langle 1\rangle 1$. $LL1Refinement \Rightarrow LL1NVRAMHistorySummaryUncorrupted = HLAlive$

$\langle 2\rangle 1$. HAVE $LL1Refinement$

$\langle 2\rangle 2$. $LL1NVRAMHistorySummaryUncorrupted \in$ BOOLEAN

BY DEF $LL1NVRAMHistorySummaryUncorrupted$

$\langle 2\rangle 3$. $HLAlive \in$ BOOLEAN

BY $\langle 2\rangle 1$ DEF $LL1Refinement$

$\langle 2\rangle 4$. IF $LL1NVRAMHistorySummaryUncorrupted$ THEN $HLAlive =$ TRUE ELSE $HLAlive =$ FALSE

BY $\langle 2\rangle 1$ DEF $LL1Refinement$

$\langle 2\rangle 5$. QED

BY $\langle 2\rangle 2$, $\langle 2\rangle 3$, $\langle 2\rangle 4$

$\langle 1\rangle 2$. $LL1Refinement' \Rightarrow LL1NVRAMHistorySummaryUncorrupted' = HLAlive'$

$\langle 2\rangle 1$. HAVE $LL1Refinement'$

$\langle 2\rangle 2$. $LL1NVRAMHistorySummaryUncorrupted' \in$ BOOLEAN

BY DEF $LL1NVRAMHistorySummaryUncorrupted$

$\langle 2\rangle 3$. $HLAlive' \in$ BOOLEAN

BY $\langle 2\rangle 1$ DEF $LL1Refinement$

$\langle 2\rangle 4$. IF $LL1NVRAMHistorySummaryUncorrupted'$ THEN $HLAlive' =$ TRUE ELSE $HLAlive' =$ FALSE

BY $\langle 2\rangle 1$ DEF $LL1Refinement$

$\langle 2\rangle 5$. QED

BY $\langle 2\rangle 2$, $\langle 2\rangle 3$, $\langle 2\rangle 4$

$\langle 1\rangle 3$. QED

BY $\langle 1\rangle 1$, $\langle 1\rangle 2$

The *LL1Implementation* theorem is where the rubber meets the road. This is the ultimate proof that the Memoir-Basic spec implements the high-level spec, under the defined refinement.

THEOREM $LL1Implementation \triangleq LL1Spec \wedge \Box LL1Refinement \Rightarrow HLSpec$

This proof will require the *LL1TypeInvariant* and the *CorrectnessInvariants*. Fortunately, the *LL1TypeSafe* theorem has already proven that the Memoir-Basic spec satisfies its type invariant, and the *CorrectnessInvariance* theorem has already proven that the Memoir-Basic spec satisfies the *CorrectnessInvariant*.

$\langle 1\rangle 1$. $LL1Spec \Rightarrow \Box LL1TypeInvariant$

BY $LL1TypeSafe$

$\langle 1\rangle 2$. $LL1Spec \Rightarrow \Box CorrectnessInvariants$

BY $LL1TypeSafe$, $CorrectnessInvariance$

The top level of the proof is boilerplate TLA+ for a *StepSimulation* proof. First, we prove that the initial predicate of the Memoir-Basic spec, conjoined with the *LL1Refinement* and invariants, implies the initial predicate of the high-level spec. Second, we prove that the *LL1Next* predicate, conjoined with the *LL1Refinement* and invariants in both primed and unprimed states, implies the *HLNext* pridicate. Third, we use temporal induction to prove that these two conditions imply that, if the *LL1Refinement* and the invariants always hold, the *LL1Spec* implies the *HLSpec*.

$\langle 1\rangle 3$. $LL1Init \wedge LL1Refinement \wedge LL1TypeInvariant \wedge CorrectnessInvariants \Rightarrow HLInit$

We begin the base case by assuming the antecedent.

$\langle 2\rangle 1.$ HAVE $LL1Init \land LL1Refinement \land LL1TypeInvariant \land CorrectnessInvariants$

$\langle 2\rangle 2.$ $LL1TypeInvariant$

BY $\langle 2\rangle 1$

We pick a *symmetricKey* that satisfies the *LL1Init* predicate.

$\langle 2\rangle 3.$ PICK $symmetricKey \in SymmetricKeyType : LL1Init!(symmetricKey)!1$

BY $\langle 2\rangle 1$ DEF $LL1Init$

We re-state the definitions from the *LL1Refinement*.

$\langle 2\rangle$ $refPrivateStateEnc \triangleq SymmetricEncrypt(LL1NVRAM.symmetricKey, HLPrivateState)$

$\langle 2\rangle$ $refStateHash \triangleq Hash(HLPublicState, refPrivateStateEnc)$

$\langle 2\rangle$ $refHistoryStateBinding \triangleq Hash(LL1NVRAM.historySummary, refStateHash)$

We prove that the definitions from the *LL1Refinement* satisfy their types, using the *LL1RefinementDefsTypeSafeLemma*.

$\langle 2\rangle 4.$ $\land$ $refPrivateStateEnc \in PrivateStateEncType$
$\land$ $refStateHash \in HashType$
$\land$ $refHistoryStateBinding \in HashType$

BY $\langle 2\rangle 1$, $\langle 2\rangle 2$, $LL1RefinementDefsTypeSafeLemma$

We hide the definitions from the *LL1Refinement*.

$\langle 2\rangle$ HIDE DEF $refPrivateStateEnc$, $refStateHash$, $refHistoryStateBinding$

We re-state the definitions from *LL1Init*.

$\langle 2\rangle$ $initialPrivateStateEnc \triangleq SymmetricEncrypt(symmetricKey, InitialPrivateState)$

$\langle 2\rangle$ $initialStateHash \triangleq Hash(InitialPublicState, initialPrivateStateEnc)$

$\langle 2\rangle$ $initialHistoryStateBinding \triangleq Hash(BaseHashValue, initialStateHash)$

$\langle 2\rangle$ $initialAuthenticator \triangleq GenerateMAC(symmetricKey, initialHistoryStateBinding)$

$\langle 2\rangle$ $initialUntrustedStorage \triangleq [$
$publicState \mapsto InitialPublicState,$
$privateStateEnc \mapsto initialPrivateStateEnc,$
$historySummary \mapsto BaseHashValue,$
$authenticator \mapsto initialAuthenticator]$

$\langle 2\rangle$ $initialTrustedStorage \triangleq [$
$historySummary \mapsto BaseHashValue,$
$symmetricKey \mapsto symmetricKey]$

We prove that the definitions from *LL1Init* satisfy their types, using the *LL1InitDefsTypeSafeLemma*.

$\langle 2\rangle 5.$ $\land$ $initialPrivateStateEnc \in PrivateStateEncType$
$\land$ $initialStateHash \in HashType$
$\land$ $initialHistoryStateBinding \in HashType$
$\land$ $initialAuthenticator \in MACType$
$\land$ $initialUntrustedStorage \in LL1UntrustedStorageType$
$\land$ $initialTrustedStorage \in LL1TrustedStorageType$

$\langle 3\rangle 1.$ $symmetricKey \in SymmetricKeyType$

BY $\langle 2\rangle 3$

$\langle 3\rangle 2.$ QED

BY $\langle 2\rangle 2$, $\langle 3\rangle 1$, $LL1InitDefsTypeSafeLemma$

We hide the definitions from *LL1Init*.

$\langle 2\rangle$ HIDE DEF $initialPrivateStateEnc$, $initialStateHash$, $initialHistoryStateBinding$, $initialAuthenticator$, $initialUntrustedStorage$, $initialTrustedStorage$

A couple of the steps below require knowing that the initial history state binding is authenticated.

$\langle 2\rangle 6.$ $LL1HistoryStateBindingAuthenticated(initialHistoryStateBinding)$

The initial authenticator was genererated as a *MAC* of the initial history state binding by *LL1Init*, using a symmetric key that matches the symmetric key in the *NVRAM*.

$\langle 3\rangle 1.$ $initialAuthenticator = GenerateMAC(LL1NVRAM.symmetricKey, initialHistoryStateBinding)$

$\langle 4\rangle 1.$ $LL1NVRAM.symmetricKey = symmetricKey$

$\langle 5\rangle 1.$ $LL1NVRAM = [historySummary \mapsto BaseHashValue,$
$symmetricKey \mapsto symmetricKey]$

BY $\langle 2\rangle 3$

$\langle 5\rangle 2.$ QED

BY $\langle 5\rangle 1$

$\langle 4\rangle 2.$ QED

BY $\langle 2\rangle 3$, $\langle 4\rangle 1$

DEF $initialAuthenticator$, $initialHistoryStateBinding$,
$initialStateHash$, $initialPrivateStateEnc$

We can thus use the $MACComplete$ property to show that the generated MAC validates appropriately. To do this, we first need to prove some types.

$\langle 3\rangle 2.$ $LL1NVRAM.symmetricKey \in SymmetricKeyType$

BY $\langle 2\rangle 2$, $LL1SubtypeImplicationLemma$DEF $LL1SubtypeImplication$

$\langle 3\rangle 3.$ $initialHistoryStateBinding \in HashType$

BY $\langle 2\rangle 5$

Then, we appeal to the $MACComplete$ property in a straightforward way.

$\langle 3\rangle 4.$ $ValidateMAC(LL1NVRAM.symmetricKey, initialHistoryStateBinding, initialAuthenticator)$

BY $\langle 3\rangle 1$, $\langle 3\rangle 2$, $\langle 3\rangle 3$, $MACComplete$

$\langle 3\rangle 5.$ $initialAuthenticator \in LL1ObservedAuthenticators$

$\langle 4\rangle 1.$ $LL1ObservedAuthenticators = \{initialAuthenticator\}$

BY $\langle 2\rangle 3$

DEF $initialAuthenticator$, $initialHistoryStateBinding$,
$initialStateHash$, $initialPrivateStateEnc$

$\langle 4\rangle 2.$ QED

BY $\langle 4\rangle 1$

$\langle 3\rangle 6.$ QED

BY $\langle 3\rangle 4$, $\langle 3\rangle 5$ DEF $LL1HistoryStateBindingAuthenticated$

For a couple of the steps below, we will need to know that the history state binding in the $NVRAM$ is true.

$\langle 2\rangle 7.$ $LL1NVRAMHistorySummaryUncorrupted$

The definition of $LL1NVRAMHistorySummaryUncorrupted$ asserts that there is some state hash bound to the history summary in the $NVRAM$ by a history state binding that is authenticated. Our witness for the state hash is the initial state hash.

$\langle 3\rangle 1.$ $initialStateHash \in HashType$

BY $\langle 2\rangle 5$

The initial state hash is bound to the base hash value by the initial history state binding, and the base hash value is the initial value of the history summary in the $NVRAM$.

$\langle 3\rangle 2.$ $LL1NVRAM.historySummary = BaseHashValue$

$\langle 4\rangle 1.$ $LL1NVRAM = [historySummary \mapsto BaseHashValue,$
$symmetricKey \mapsto symmetricKey]$

BY $\langle 2\rangle 3$

$\langle 4\rangle 2.$ QED

BY $\langle 4\rangle 1$

The initial history state binding is authenticated.

$\langle 3\rangle 3.$ $LL1HistoryStateBindingAuthenticated(initialHistoryStateBinding)$

BY $\langle 2\rangle 6$

$\langle 3\rangle 4.$ QED

BY $\langle 3\rangle 1$, $\langle 3\rangle 2$, $\langle 3\rangle 3$ DEF $initialHistoryStateBinding$, $LL1NVRAMHistorySummaryUncorrupted$

The bulk of the proof for the initial case is proving that the public and private state defined by the refinememt match the initial public and private state.

$\langle 2 \rangle 8.\ \land\ \mathit{HLPublicState} = \mathit{InitialPublicState}$
$\land\ \mathit{HLPrivateState} = \mathit{InitialPrivateState}$

We show that the refined public state matches the initial public state that the refined encrypted private state matches the encrypted initial state. We will use the *HashCollisionResistant* property.

$\langle 3 \rangle 1.\ \land\ \mathit{HLPublicState} = \mathit{InitialPublicState}$
$\land\ \mathit{refPrivateStateEnc} = \mathit{initialPrivateStateEnc}$

We begin by proving some types we will need to appeal to the *HashCollisionResistant* property.

$\langle 4 \rangle 1.\ \mathit{HLPublicState} \in \mathit{HashDomain}$
$\langle 5 \rangle 1.\ \mathit{HLPublicState} \in \mathit{PublicStateType}$
BY $\langle 2 \rangle 1$, $\langle 2 \rangle 7$ DEF $\mathit{LL1Refinement}$
$\langle 5 \rangle 2.$ QED
BY $\langle 5 \rangle 1$ DEF $\mathit{HashDomain}$
$\langle 4 \rangle 2.\ \mathit{InitialPublicState} \in \mathit{HashDomain}$
$\langle 5 \rangle 1.\ \mathit{InitialPublicState} \in \mathit{PublicStateType}$
BY $\mathit{ConstantsTypeSafe}$
$\langle 5 \rangle 2.$ QED
BY $\langle 5 \rangle 1$ DEF $\mathit{HashDomain}$
$\langle 4 \rangle 3.\ \mathit{refPrivateStateEnc} \in \mathit{HashDomain}$
$\langle 5 \rangle 1.\ \mathit{refPrivateStateEnc} \in \mathit{PrivateStateEncType}$
BY $\langle 2 \rangle 4$
$\langle 5 \rangle 2.$ QED
BY $\langle 5 \rangle 1$ DEF $\mathit{HashDomain}$
$\langle 4 \rangle 4.\ \mathit{initialPrivateStateEnc} \in \mathit{HashDomain}$
$\langle 5 \rangle 1.\ \mathit{initialPrivateStateEnc} \in \mathit{PrivateStateEncType}$
BY $\langle 2 \rangle 5$
$\langle 5 \rangle 2.$ QED
BY $\langle 5 \rangle 1$ DEF $\mathit{HashDomain}$

We then prove that the refined state hash equals the initial state hash. We will use the *UniquenessInvariant*.

$\langle 4 \rangle 5.\ \mathit{refStateHash} = \mathit{initialStateHash}$

We prove some types we will need for the *UniquenessInvariant*.

$\langle 5 \rangle 1.\ \mathit{refStateHash} \in \mathit{HashType}$
BY $\langle 2 \rangle 4$
$\langle 5 \rangle 2.\ \mathit{initialStateHash} \in \mathit{HashType}$
BY $\langle 2 \rangle 5$

We prove that the refined history state binding is authenticated. This follows directly from the *LL1Refinement*.

$\langle 5 \rangle 3.\ \mathit{LL1HistoryStateBindingAuthenticated}(\mathit{refHistoryStateBinding})$
$\langle 6 \rangle 1.\ \mathit{LL1Refinement}$
BY $\langle 2 \rangle 1$
$\langle 6 \rangle 2.\ \mathit{LL1NVRAMHistorySummaryUncorrupted}$
BY $\langle 2 \rangle 7$
$\langle 6 \rangle 3.$ QED
BY $\langle 6 \rangle 1$, $\langle 6 \rangle 2$ DEF $\mathit{LL1Refinement}$, $\mathit{refHistoryStateBinding}$, $\mathit{refStateHash}$, $\mathit{refPrivateStateEnc}$

We prove that the initial history state binding is authenticated. We will use the *MACComplete* property.

$\langle 5 \rangle 4.\ \mathit{LL1HistoryStateBindingAuthenticated}(\mathit{initialHistoryStateBinding})$
BY $\langle 2 \rangle 6$

Since both history state bindings are authenticated, it follows that the two state hashes are equal.

$\langle 5 \rangle 5.$ QED
$\langle 6 \rangle 1.\ \mathit{UniquenessInvariant}$
BY $\langle 2 \rangle 1$ DEF $\mathit{CorrectnessInvariants}$

For the *UniquenessInvariant* to apply, we have to show that the history summary in the initial history state binding is equal to the history summary in the *NVRAM*.

$\langle 6\rangle 2.\ LL1NVRAM.historySummary = BaseHashValue$

$\langle 8\rangle 1.\ LL1NVRAM = [historySummary \mapsto BaseHashValue,$
$\quad symmetricKey \mapsto symmetricKey]$

BY $\langle 2\rangle 3$

$\langle 8\rangle 2.$ QED

BY $\langle 8\rangle 1$

$\langle 6\rangle 3.$ QED

BY $\langle 5\rangle 1$, $\langle 5\rangle 2$, $\langle 5\rangle 3$, $\langle 5\rangle 4$, $\langle 6\rangle 1$, $\langle 6\rangle 2$

DEF *UniquenessInvariant*, *refHistoryStateBinding*, *initialHistoryStateBinding*

$\langle 4\rangle 6.$ QED

BY $\langle 4\rangle 1$, $\langle 4\rangle 2$, $\langle 4\rangle 3$, $\langle 4\rangle 4$, $\langle 4\rangle 5$, *HashCollisionResistant*DEF *refStateHash*, *initialStateHash*

From the previous step, it immediately follows that the refined public state matches the initial public state.

$\langle 3\rangle 2.\ HLPublicState = InitialPublicState$

BY $\langle 3\rangle 1$

It is slightly more involved to show that the refined private state matches the initial private state. We need to show that the symmetric key that satisfies the *LL1Init* predicate is the same symmetric key that is in the *NVRAM*, and we also have to employ the correctness of the symmetric crypto operations.

$\langle 3\rangle 3.\ HLPrivateState = InitialPrivateState$

We need to prove some types.

$\langle 4\rangle 1.\ LL1NVRAM.symmetricKey \in SymmetricKeyType$

BY $\langle 2\rangle 2$, *LL1SubtypeImplicationLemma*DEF *LL1SubtypeImplication*

$\langle 4\rangle 2.\ refPrivateStateEnc = initialPrivateStateEnc$

BY $\langle 3\rangle 1$

The refined private state equals the decryption of the refined encrypted private state, using the symmetry key brought into existence in *LL1Init*.

$\langle 4\rangle 3.\ HLPrivateState = SymmetricDecrypt(symmetricKey, refPrivateStateEnc)$

The refined private state equals the decryption of the refined encrypted private state, using the symmetry key in the *NVRAM*. This follows because the refined encrypted private state is defined in the *LL1Refinement* as the encryption of the refined private state, and the symmetric crypto operations are assumed to be correct.

$\langle 5\rangle 1.\ HLPrivateState = SymmetricDecrypt(LL1NVRAM.symmetricKey, refPrivateStateEnc)$

$\langle 6\rangle 1.\ HLPrivateState \in PrivateStateType$

BY $\langle 2\rangle 1$, $\langle 2\rangle 7$ DEF *LL1Refinement*

$\langle 6\rangle 2.$ QED

BY $\langle 4\rangle 1$, $\langle 6\rangle 1$, *SymmetricCryptoCorrect*DEF *refPrivateStateEnc*

The symmetric key in the *NVRAM* matches the symmetric key brought into existence in *LL1Init*.

$\langle 5\rangle 2.\ LL1NVRAM.symmetricKey = symmetricKey$

$\langle 6\rangle 1.\ LL1NVRAM = [historySummary \mapsto BaseHashValue,$
$\quad symmetricKey \mapsto symmetricKey]$

BY $\langle 2\rangle 3$

$\langle 6\rangle 2.$ QED

BY $\langle 6\rangle 1$

$\langle 5\rangle 3.$ QED

BY $\langle 5\rangle 1$, $\langle 5\rangle 2$

The initial private state equals the decryption of the initial encrypted private state, using the symmetry key brought into existence in *LL1Init*. This follows because the refined encrypted private state is defined in *LL1Init* as the encryption of the initial private state, and the symmetric crypto operations are assumed to be correct.

$\langle 4\rangle 4.\ InitialPrivateState = SymmetricDecrypt(symmetricKey, initialPrivateStateEnc)$

$\langle 5\rangle 1.\ InitialPrivateState \in PrivateStateType$

BY *ConstantsTypeSafe*

$\langle 5\rangle 2.$ QED

BY $\langle 4\rangle 1$, $\langle 5\rangle 1$, *SymmetricCryptoCorrect*DEF *initialPrivateStateEnc*

$\langle 4 \rangle 5.$ QED
BY $\langle 4 \rangle 2$, $\langle 4 \rangle 3$, $\langle 4 \rangle 4$

The truth of the conjuncts implies the truth of the conjunction.

$\langle 3 \rangle 4.$ QED
BY $\langle 3 \rangle 2$, $\langle 3 \rangle 3$

The QED step simply asserts each conjunct in the *HLinit* predicate.

$\langle 2 \rangle 9.$ QED
$\langle 3 \rangle 1.$ $HLAlive = \text{TRUE}$
$\langle 4 \rangle 1.$ $LL1NVRAMHistorySummaryUncorrupted$
BY $\langle 2 \rangle 7$
$\langle 4 \rangle 2.$ QED
BY $\langle 2 \rangle 1$, $\langle 4 \rangle 1$ DEF $LL1Refinement$
$\langle 3 \rangle 2.$ $HLAvailableInputs = InitialAvailableInputs$
$\langle 4 \rangle 1.$ $LL1AvailableInputs = InitialAvailableInputs$
BY $\langle 2 \rangle 3$
$\langle 4 \rangle 2.$ $HLAvailableInputs = LL1AvailableInputs$
BY $\langle 2 \rangle 1$ DEF $LL1Refinement$
$\langle 4 \rangle 3.$ QED
BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$
$\langle 3 \rangle 3.$ $HLObservedOutputs = \{\}$
$\langle 4 \rangle 1.$ $LL1ObservedOutputs = \{\}$
BY $\langle 2 \rangle 3$
$\langle 4 \rangle 2.$ $HLObservedOutputs = LL1ObservedOutputs$
BY $\langle 2 \rangle 1$ DEF $LL1Refinement$
$\langle 4 \rangle 3.$ QED
BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$
$\langle 3 \rangle 4.$ $HLPublicState = InitialPublicState$
BY $\langle 2 \rangle 8$
$\langle 3 \rangle 5.$ $HLPrivateState = InitialPrivateState$
BY $\langle 2 \rangle 8$
$\langle 3 \rangle 6.$ QED
BY $\langle 3 \rangle 1$, $\langle 3 \rangle 2$, $\langle 3 \rangle 3$, $\langle 3 \rangle 4$, $\langle 3 \rangle 5$ DEF $HLInit$

For the induction step, we will need the refinement, the type invariant, and the correctness invariants to be true in both the unprimed and primed states.

$\langle 1 \rangle 4.$ $(\land\ [LL1Next]_{LL1Vars}$
$\land\ LL1Refinement$
$\land\ LL1Refinement'$
$\land\ LL1TypeInvariant$
$\land\ LL1TypeInvariant'$
$\land\ CorrectnessInvariants$
$\land\ CorrectnessInvariants')$
$\Rightarrow$
$[HLNext]_{HLVars}$

We assume the antecedents.

$\langle 2 \rangle 1.$ HAVE $\land\ [LL1Next]_{LL1Vars}$
$\land\ LL1Refinement$
$\land\ LL1Refinement'$
$\land\ LL1TypeInvariant$
$\land\ LL1TypeInvariant'$
$\land\ CorrectnessInvariants$
$\land\ CorrectnessInvariants'$

We then prove that each step in the Memoir-Basic spec refines to a step in the high-level spec. First, a Memoir-Basic stuttering step refines to a high-level stuttering step.

$\langle 2 \rangle 2.$ UNCHANGED $LL1Vars \Rightarrow$ UNCHANGED $HLVars$

$\langle 3 \rangle 1.$ HAVE UNCHANGED $LL1Vars$

The *HLAlive* predicate is unchanged because the *LL1NVRAMHistorySummaryUncorrupted* predicate is unchanged.

$\langle 3 \rangle 2.$ UNCHANGED $HLAlive$

The *LL1NVRAMHistorySummaryUncorruptedEqualsHLAliveLemma* tells us that *LL1NVRAMHistorySummaryUncorrupted* and *HLAlive* are equal.

$\langle 4 \rangle 1.$ $LL1NVRAMHistorySummaryUncorrupted = HLAlive$

$\langle 5 \rangle 1.$ $LL1Refinement$

BY $\langle 2 \rangle 1$

$\langle 5 \rangle 2.$ QED

BY $\langle 5 \rangle 1$, $LL1NVRAMHistorySummaryUncorruptedEqualsHLAliveLemma$

$\langle 4 \rangle 2.$ $LL1NVRAMHistorySummaryUncorrupted' = HLAlive'$

$\langle 5 \rangle 1.$ $LL1Refinement'$

BY $\langle 2 \rangle 1$

$\langle 5 \rangle 2.$ QED

BY $\langle 5 \rangle 1$, $LL1NVRAMHistorySummaryUncorruptedEqualsHLAliveLemma$

Then, the *LL1NVRAMHistorySummaryUncorruptedUnchangedLemma* tells us that the *LL1NVRAMHistorySummaryUncorrupted* predicate is unchanged.

$\langle 4 \rangle 3.$ UNCHANGED $LL1NVRAMHistorySummaryUncorrupted$

$\langle 5 \rangle 1.$ $LL1TypeInvariant$

BY $\langle 2 \rangle 1$

$\langle 5 \rangle 2.$ UNCHANGED $LL1NVRAM$

BY $\langle 3 \rangle 1$ DEF $LL1Vars$

$\langle 5 \rangle 3.$ $\forall\, historyStateBinding \in HashType :$
UNCHANGED $LL1HistoryStateBindingAuthenticated(historyStateBinding)$

$\langle 6 \rangle 1.$ UNCHANGED $\langle LL1NVRAM, LL1ObservedAuthenticators \rangle$

BY $\langle 3 \rangle 1$ DEF $LL1Vars$

$\langle 6 \rangle 2.$ QED

BY $\langle 6 \rangle 1$, $UnchangedAuthenticatedHistoryStateBindingsLemma$

$\langle 5 \rangle 4.$ QED

BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$, $\langle 5 \rangle 3$, $LL1NVRAMHistorySummaryUncorruptedUnchangedLemma$

$\langle 4 \rangle 4.$ QED

BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$, $\langle 4 \rangle 3$

The mapping from *LL1AvailableInputs* to *HLAvailableInputs* is direct.

$\langle 3 \rangle 3.$ UNCHANGED $HLAvailableInputs$

$\langle 4 \rangle 1.$ UNCHANGED $LL1AvailableInputs$

BY $\langle 3 \rangle 1$ DEF $LL1Vars$

$\langle 4 \rangle 2.$ $\land\; HLAvailableInputs = LL1AvailableInputs$
$\land\; HLAvailableInputs' = LL1AvailableInputs'$

BY $\langle 2 \rangle 1$ DEF $LL1Refinement$

$\langle 4 \rangle 3.$ QED

BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$

The mapping from *LL1ObservedOutputs* to *HLObservedOutputs* is direct.

$\langle 3 \rangle 4.$ UNCHANGED $HLObservedOutputs$

$\langle 4 \rangle 1.$ UNCHANGED $LL1ObservedOutputs$

BY $\langle 3 \rangle 1$ DEF $LL1Vars$

$\langle 4 \rangle 2.$ $\land\; HLObservedOutputs = LL1ObservedOutputs$
$\land\; HLObservedOutputs' = LL1ObservedOutputs'$

BY $\langle 2 \rangle 1$ DEF $LL1Refinement$

$\langle 4 \rangle 3.$ QED
BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$

We prove the stuttering of the high-level public and private state by using the *NonAdvancementLemma*.

$\langle 3 \rangle 5.$ UNCHANGED $\langle HLPublicState,\ HLPrivateState \rangle$

Many of the antecedents for the *NonAdvancementLemma* come directly from antecedents in the induction.

$\langle 4 \rangle 1.$ $\wedge$ $LL1Refinement$
$\wedge$ $LL1Refinement'$
$\wedge$ $LL1TypeInvariant$
$\wedge$ $LL1TypeInvariant'$
$\wedge$ $UniquenessInvariant$
BY $\langle 2 \rangle 1$ DEF $CorrectnessInvariants$

The *LL1NVRAM* is unchanged because the Memoir-Basic variables are unchanged.

$\langle 4 \rangle 2.$ UNCHANGED $LL1NVRAM$
BY $\langle 3 \rangle 1$ DEF $LL1Vars$

The *UnchangedAuthenticatedHistoryStateBindingsLemma* tells us that there is no change to the set of history state bindings that have authenticators in the set *LL1ObservedAuthenticators*.

$\langle 4 \rangle 3.$ $\forall\, historyStateBinding \in HashType :$
UNCHANGED $LL1HistoryStateBindingAuthenticated(historyStateBinding)$
$\langle 5 \rangle 1.$ UNCHANGED $\langle LL1NVRAM,\ LL1ObservedAuthenticators \rangle$
BY $\langle 3 \rangle 1$ DEF $LL1Vars$
$\langle 5 \rangle 2.$ QED
BY $\langle 5 \rangle 1$, $UnchangedAuthenticatedHistoryStateBindingsLemma$

We have all of the antecedents for the *NonAdvancementLemma*, so we can apply it directly.

$\langle 4 \rangle 4.$ QED
BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$, $\langle 4 \rangle 3$, $NonAdvancementLemma$
$\langle 3 \rangle 6.$ QED
BY $\langle 3 \rangle 2$, $\langle 3 \rangle 3$, $\langle 3 \rangle 4$, $\langle 3 \rangle 5$ DEF $HLVars$

A Memoir-Basic *LL1MakeInputAvailable* action refines to a high-level *HLMakeInputAvailable* action.

$\langle 2 \rangle 3.$ $LL1MakeInputAvailable \Rightarrow HLMakeInputAvailable$
$\langle 3 \rangle 1.$ HAVE $LL1MakeInputAvailable$
$\langle 3 \rangle 2.$ PICK $input \in InputType : LL1MakeInputAvailable!(input)$
BY $\langle 3 \rangle 1$ DEF $LL1MakeInputAvailable$

The mapping from *LL1AvailableInputs* to *HLAvailableInputs* is direct.

$\langle 3 \rangle 3.$ $input \notin HLAvailableInputs$
$\langle 4 \rangle 1.$ $input \notin LL1AvailableInputs$
BY $\langle 3 \rangle 2$
$\langle 4 \rangle 2.$ $\wedge\ HLAvailableInputs = LL1AvailableInputs$
$\wedge\ HLAvailableInputs' = LL1AvailableInputs'$
BY $\langle 2 \rangle 1$ DEF $LL1Refinement$
$\langle 4 \rangle 3.$ QED
BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$
$\langle 3 \rangle 4.$ $HLAvailableInputs' = HLAvailableInputs \cup \{input\}$
$\langle 4 \rangle 1.$ $LL1AvailableInputs' = LL1AvailableInputs \cup \{input\}$
BY $\langle 3 \rangle 2$
$\langle 4 \rangle 2.$ $\wedge\ HLAvailableInputs = LL1AvailableInputs$
$\wedge\ HLAvailableInputs' = LL1AvailableInputs'$
BY $\langle 2 \rangle 1$ DEF $LL1Refinement$
$\langle 4 \rangle 3.$ QED
BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$

The mapping from *LL1ObservedOutputs* to *HLObservedOutputs* is direct.

$\langle 3 \rangle 5$. UNCHANGED *HLObservedOutputs*

$\langle 4 \rangle 1$. UNCHANGED *LL1ObservedOutputs*

BY $\langle 3 \rangle 2$

$\langle 4 \rangle 2$. $\land$ *HLObservedOutputs* = *LL1ObservedOutputs*
$\land$ *HLObservedOutputs'* = *LL1ObservedOutputs'*

BY $\langle 2 \rangle 1$ DEF *LL1Refinement*

$\langle 4 \rangle 3$. QED

BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$

The *HLAlive* predicate is unchanged because the *LL1NVRAMHistorySummaryUncorrupted* predicate is unchanged.

$\langle 3 \rangle 6$. UNCHANGED *HLAlive*

The *LL1NVRAMHistorySummaryUncorruptedEqualsHLAliveLemma* tells us that *LL1NVRAMHistorySummaryUncorrupted* and *HLAlive* are equal.

$\langle 4 \rangle 1$. *LL1NVRAMHistorySummaryUncorrupted* = *HLAlive*

$\langle 5 \rangle 1$. *LL1Refinement*

BY $\langle 2 \rangle 1$

$\langle 5 \rangle 2$. QED

BY $\langle 5 \rangle 1$, *LL1NVRAMHistorySummaryUncorruptedEqualsHLAliveLemma*

$\langle 4 \rangle 2$. *LL1NVRAMHistorySummaryUncorrupted'* = *HLAlive'*

$\langle 5 \rangle 1$. *LL1Refinement'*

BY $\langle 2 \rangle 1$

$\langle 5 \rangle 2$. QED

BY $\langle 5 \rangle 1$, *LL1NVRAMHistorySummaryUncorruptedEqualsHLAliveLemma*

Then, the *LL1NVRAMHistorySummaryUncorruptedUnchangedLemma* tells us that the *LL1NVRAMHistorySummaryUncorrupted* predicate is unchanged.

$\langle 4 \rangle 3$. UNCHANGED *LL1NVRAMHistorySummaryUncorrupted*

$\langle 5 \rangle 1$. *LL1TypeInvariant*

BY $\langle 2 \rangle 1$

$\langle 5 \rangle 2$. UNCHANGED *LL1NVRAM*

BY $\langle 3 \rangle 2$

$\langle 5 \rangle 3$. $\forall$ *historyStateBinding* $\in$ *HashType* :
UNCHANGED *LL1HistoryStateBindingAuthenticated*(*historyStateBinding*)

$\langle 6 \rangle 1$. UNCHANGED $\langle$*LL1NVRAM*, *LL1ObservedAuthenticators*$\rangle$

BY $\langle 3 \rangle 2$

$\langle 6 \rangle 2$. QED

BY $\langle 6 \rangle 1$, *UnchangedAuthenticatedHistoryStateBindingsLemma*

$\langle 5 \rangle 4$. QED

BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$, $\langle 5 \rangle 3$, *LL1NVRAMHistorySummaryUncorruptedUnchangedLemma*

$\langle 4 \rangle 4$. QED

BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$, $\langle 4 \rangle 3$

We prove the stuttering of the high-level public and private state by using the *NonAdvancementLemma*.

$\langle 3 \rangle 7$. UNCHANGED $\langle$*HLPublicState*, *HLPrivateState*$\rangle$

Many of the antecedents for the *NonAdvancementLemma* come directly from antecedents in the induction.

$\langle 4 \rangle 1$. $\land$ *LL1Refinement*
$\land$ *LL1Refinement'*
$\land$ *LL1TypeInvariant*
$\land$ *LL1TypeInvariant'*
$\land$ *UniquenessInvariant*

BY $\langle 2 \rangle 1$ DEF *CorrectnessInvariants*

The *NVRAM* is unchanged by definition of the *LL1MakeInputAvailable* action.

$\langle 4 \rangle 2$. UNCHANGED *LL1NVRAM*

BY $\langle 3 \rangle 2$

The *UnchangedAuthenticatedHistoryStateBindingsLemma* tells us that there is no change to the set of history state bindings that have authenticators in the set *LL1ObservedAuthenticators*.

$\langle 4\rangle 3.$ $\forall$ *historyStateBinding* $\in$ *HashType* :
 UNCHANGED *LL1HistoryStateBindingAuthenticated*(*historyStateBinding*)

$\langle 5\rangle 1.$ UNCHANGED $\langle$*LL1NVRAM*, *LL1ObservedAuthenticators*$\rangle$
 BY $\langle 3\rangle 2$

$\langle 5\rangle 2.$ QED
 BY $\langle 5\rangle 1$, *UnchangedAuthenticatedHistoryStateBindingsLemma*

We have all of the antecedents for the *NonAdvancementLemma*, so we can apply it directly.

$\langle 4\rangle 4.$ QED
 BY $\langle 4\rangle 1$, $\langle 4\rangle 2$, $\langle 4\rangle 3$, *NonAdvancementLemma*

$\langle 3\rangle 8.$ QED
 BY $\langle 3\rangle 2$, $\langle 3\rangle 3$, $\langle 3\rangle 4$, $\langle 3\rangle 5$, $\langle 3\rangle 6$, $\langle 3\rangle 7$ DEF *HLMakeInputAvailable*

A Memoir-Basic *LL1PerformOperation* action refines to a high-level *HLAdvanceService* action.

$\langle 2\rangle 4.$ *LL1PerformOperation* $\Rightarrow$ *HLAdvanceService*

We assume the antecedent.

$\langle 3\rangle 1.$ HAVE *LL1PerformOperation*

We pick an input that satisfies the *LL1PerformOperation* predicate.

$\langle 3\rangle 2.$ PICK *input* $\in$ *LL1AvailableInputs* : *LL1PerformOperation*!(*input*)!1
 BY $\langle 3\rangle 1$ DEF *LL1PerformOperation*

We re-state the definitions from the *LL1Refinement*

$\langle 3\rangle$ *refPrivateStateEnc* $\triangleq$ *SymmetricEncrypt*(*LL1NVRAM*.*symmetricKey*, *HLPrivateState*)
$\langle 3\rangle$ *refStateHash* $\triangleq$ *Hash*(*HLPublicState*, *refPrivateStateEnc*)
$\langle 3\rangle$ *refHistoryStateBinding* $\triangleq$ *Hash*(*LL1NVRAM*.*historySummary*, *refStateHash*)

We prove that the definitions from the *LL1Refinement* satisfy their types, using the *LL1RefinementDefsTypeSafeLemma*.

$\langle 3\rangle 3.$ $\land$ *refPrivateStateEnc* $\in$ *PrivateStateEncType*
 $\land$ *refStateHash* $\in$ *HashType*
 $\land$ *refHistoryStateBinding* $\in$ *HashType*

$\langle 4\rangle 1.$ *LL1Refinement*
 BY $\langle 2\rangle 1$

$\langle 4\rangle 2.$ *LL1TypeInvariant*
 BY $\langle 2\rangle 1$

$\langle 4\rangle 3.$ QED
 BY $\langle 4\rangle 1$, $\langle 4\rangle 2$, *LL1RefinementDefsTypeSafeLemma*

We also prove that the primed definitions from the *LL1Refinement* satisfy their types, using the *LL1RefinementPrimeDefsTypeSafeLemma*.

$\langle 3\rangle 4.$ $\land$ *refPrivateStateEnc*$'$ $\in$ *PrivateStateEncType*
 $\land$ *refStateHash*$'$ $\in$ *HashType*
 $\land$ *refHistoryStateBinding*$'$ $\in$ *HashType*

$\langle 4\rangle 1.$ *LL1Refinement*$'$
 BY $\langle 2\rangle 1$

$\langle 4\rangle 2.$ *LL1TypeInvariant*$'$
 BY $\langle 2\rangle 1$

$\langle 4\rangle 3.$ QED
 BY $\langle 4\rangle 1$, $\langle 4\rangle 2$, *LL1RefinementPrimeDefsTypeSafeLemma*

We hide the definitions from the *LL1Refinement*.

$\langle 3\rangle$ HIDE DEF *refPrivateStateEnc*, *refStateHash*, *refHistoryStateBinding*

We re-state the definitions from *LL1PerformOperation*.

$\langle 3\rangle$ *stateHash* $\triangleq$ *Hash*(*LL1RAM*.*publicState*, *LL1RAM*.*privateStateEnc*)

⟨3⟩ *historyStateBinding* $\triangleq$ *Hash*(*LL1RAM.historySummary*, *stateHash*)
⟨3⟩ *privateState* $\triangleq$ *SymmetricDecrypt*(*LL1NVRAM.symmetricKey*, *LL1RAM.privateStateEnc*)
⟨3⟩ *sResult* $\triangleq$ *Service*(*LL1RAM.publicState*, *privateState*, *input*)
⟨3⟩ *newPrivateStateEnc* $\triangleq$
    SymmetricEncrypt(*LL1NVRAM.symmetricKey*, *sResult.newPrivateState*)
⟨3⟩ *newHistorySummary* $\triangleq$ *Hash*(*LL1NVRAM.historySummary*, *input*)
⟨3⟩ *newStateHash* $\triangleq$ *Hash*(*sResult.newPublicState*, *newPrivateStateEnc*)
⟨3⟩ *newHistoryStateBinding* $\triangleq$ *Hash*(*newHistorySummary*, *newStateHash*)
⟨3⟩ *newAuthenticator* $\triangleq$ *GenerateMAC*(*LL1NVRAM.symmetricKey*, *newHistoryStateBinding*)

We prove that the definitions from *LL1PerformOperation* satisfy their types, using the *LL1PerformOperationDefsTypeSafeLemma*.

⟨3⟩5. ∧ *stateHash* ∈ *HashType*
  ∧ *historyStateBinding* ∈ *HashType*
  ∧ *privateState* ∈ *PrivateStateType*
  ∧ *sResult* ∈ *ServiceResultType*
  ∧ *sResult.newPublicState* ∈ *PublicStateType*
  ∧ *sResult.newPrivateState* ∈ *PrivateStateType*
  ∧ *sResult.output* ∈ *OutputType*
  ∧ *newPrivateStateEnc* ∈ *PrivateStateEncType*
  ∧ *newHistorySummary* ∈ *HashType*
  ∧ *newStateHash* ∈ *HashType*
  ∧ *newHistoryStateBinding* ∈ *HashType*
  ∧ *newAuthenticator* ∈ *MACType*
 ⟨4⟩1. *input* ∈ *LL1AvailableInputs*
  BY ⟨3⟩2
 ⟨4⟩2. *LL1TypeInvariant*
  BY ⟨2⟩1
 ⟨4⟩3. QED
  BY ⟨4⟩1, ⟨4⟩2, *LL1PerformOperationDefsTypeSafeLemma*

We hide the definitions from *LL1PerformOperation*.

⟨3⟩ HIDE DEF *stateHash*, *historyStateBinding*, *privateState*, *sResult*, *newPrivateStateEnc*,
    newHistorySummary, *newStateHash*, *newHistoryStateBinding*, *newAuthenticator*

We re-state the definition from *HLAdvanceService*.

⟨3⟩ *hlSResult* $\triangleq$ *Service*(*HLPublicState*, *HLPrivateState*, *input*)

We hide the definition from *HLAdvanceService*.

⟨3⟩ HIDE DEF *hlSResult*

We prove that the history summary in the *NVRAM* is uncorrupted in the unprimed state. This follows from the enablement conditions of the *LL1PerformOperation* action. We will expand the definitions of *LL1NVRAMHistorySummaryUncorrupted* and *LL1HistoryStateBindingAuthenticated*, and prove that the required conditions are all satisfied.

⟨3⟩6. *LL1NVRAMHistorySummaryUncorrupted*

The authenticator in the RAM authenticates the history state binding, since this is an enablement condition of the *LL1PerformOperation* action.

 ⟨4⟩1. *ValidateMAC*(*LL1NVRAM.symmetricKey*, *historyStateBinding*, *LL1RAM.authenticator*)
  BY ⟨3⟩2 DEF *historyStateBinding*, *stateHash*

The state hash defined in the *LL1PerformOperation* is in *HashType*.

 ⟨4⟩2. *stateHash* ∈ *HashType*
  BY ⟨3⟩5

From the *UnforgeabilityInvariant*, the authenticator in the RAM is in the set of observed authenticators.

 ⟨4⟩3. *LL1RAM.authenticator* ∈ *LL1ObservedAuthenticators*
  ⟨5⟩1. *UnforgeabilityInvariant*

BY $\langle 2\rangle 1$ DEF *CorrectnessInvariants*

$\langle 5\rangle 2.$ $historyStateBinding \in HashType$

BY $\langle 3\rangle 5$

$\langle 5\rangle 3.$ QED

BY $\langle 4\rangle 1$, $\langle 5\rangle 1$, $\langle 5\rangle 2$ DEF *UnforgeabilityInvariant*

The above three conditions are sufficient to satisfy the *LL1NVRAMHistorySummaryUncorrupted* predicate in the unprimed state, given that the history summary in the RAM equals the history summary in the *NVRAM*.

$\langle 4\rangle 4.$ $LL1NVRAM.historySummary = LL1RAM.historySummary$

BY $\langle 3\rangle 2$

$\langle 4\rangle 5.$ QED

BY $\langle 4\rangle 1$, $\langle 4\rangle 2$, $\langle 4\rangle 3$, $\langle 4\rangle 4$

DEF *LL1NVRAMHistorySummaryUncorrupted*, *LL1HistoryStateBindingAuthenticated*, *historyStateBinding*

We prove that the history summary in the *NVRAM* is uncorrupted in the primed state. This follows from the enablement conditions of the *LL1PerformOperation* action. We will expand the definitions of *LL1NVRAMHistorySummaryUncorrupted* and *LL1HistoryStateBindingAuthenticated*, and prove that the required conditions are all satisfied.

$\langle 3\rangle 7.$ $LL1NVRAMHistorySummaryUncorrupted'$

The new authenticator defined by the *LL1PerformOperation* action authenticates the new history state binding defined by this action. We will use the *MACComplete* property.

$\langle 4\rangle 1.$ $ValidateMAC(LL1NVRAM.symmetricKey', newHistoryStateBinding, newAuthenticator)$

The new authenticator was genererated as a *MAC* of the new history state binding by *LL1PerformOperation*, using the unchanged symmetric key in the *NVRAM*.

$\langle 5\rangle 1.$ $newAuthenticator = GenerateMAC(LL1NVRAM.symmetricKey', newHistoryStateBinding)$

$\langle 6\rangle 1.$ UNCHANGED $LL1NVRAM.symmetricKey$

BY $\langle 2\rangle 1$, *SymmetricKeyConstantLemma*

$\langle 6\rangle 2.$ QED

BY $\langle 3\rangle 2$, $\langle 6\rangle 1$

DEF *newAuthenticator*, *newHistoryStateBinding*, *newStateHash*, *newPrivateStateEnc*, *sResult*, *privateState*

We can thus use the *MACComplete* property to show that the generated *MAC* validates appropriately. To do this, we first need to prove some types.

$\langle 5\rangle 2.$ $LL1NVRAM.symmetricKey' \in SymmetricKeyType$

$\langle 6\rangle 1.$ $LL1TypeInvariant'$

BY $\langle 2\rangle 1$

$\langle 6\rangle 2.$ QED

BY $\langle 6\rangle 1$, *LL1SubtypeImplicationLemma*DEF *LL1SubtypeImplication*

$\langle 5\rangle 3.$ $newHistoryStateBinding \in HashType$

BY $\langle 3\rangle 5$

Then, we appeal to the *MACComplete* property in a straightforward way.

$\langle 5\rangle 4.$ QED

BY $\langle 5\rangle 1$, $\langle 5\rangle 2$, $\langle 5\rangle 3$, *MACComplete*

The new state hash defined by the *LL1PerformOperation* is in *HashType*.

$\langle 4\rangle 2.$ $newStateHash \in HashType$

BY $\langle 3\rangle 5$

The new authenticator defined by the *LL1PerformOperation* action is in the primed set of observed authenticators, as specified by the *LL1PerformOperation* action.

$\langle 4\rangle 3.$ $newAuthenticator \in LL1ObservedAuthenticators'$

$\langle 5\rangle 1.$ $LL1ObservedAuthenticators' = LL1ObservedAuthenticators \cup \{newAuthenticator\}$

BY $\langle 3\rangle 2$

DEF *newAuthenticator*, *newHistoryStateBinding*, *newStateHash*, *newHistorySummary*, *newPrivateStateEnc*, *sResult*, *privateState*

$\langle 5\rangle 2.$ QED

BY $\langle 5\rangle 1$

The history summary in the primed state of the *NVRAM* equals the new history summary defined by the *LL1PerformOperation*.

$\langle 4\rangle 4.$ $LL1NVRAM.historySummary' = newHistorySummary$

$\langle 5\rangle 1.$ $LL1NVRAM' = [$
$historySummary \mapsto newHistorySummary,$
$symmetricKey \mapsto LL1NVRAM.symmetricKey]$

BY $\langle 3\rangle 2$ DEF *newHistorySummary*

$\langle 5\rangle 2.$ QED

BY $\langle 5\rangle 1$

The above three conditions and the equality are sufficient to satisfy the *LL1NVRAMHistorySummaryUncorrupted* predicate in the primed state.

$\langle 4\rangle 5.$ QED

BY $\langle 4\rangle 1$, $\langle 4\rangle 2$, $\langle 4\rangle 3$, $\langle 4\rangle 4$

DEF *LL1NVRAMHistorySummaryUncorrupted*, *LL1HistoryStateBindingAuthenticated*, *newHistoryStateBinding*

The proof proper has two main steps. The first main step is to prove that the public and private states that are arguments to the *Service* in *LL1PerformOperation* are identical to the values of *HLPublicState* and *HLPrivateState* in the *LL1Refinement*.

$\langle 3\rangle 8.$ $\land$ $HLPublicState = LL1RAM.publicState$
$\land$ $HLPrivateState = privateState$

We show that the refined public state matches the public state in the RAM and that the refined encrypted private state matches the encrypted private state in the RAM. We will use the *HashCollisionResistant* property.

$\langle 4\rangle 1.$ $\land$ $HLPublicState = LL1RAM.publicState$
$\land$ $refPrivateStateEnc = LL1RAM.privateStateEnc$

We begin by proving some types we will need to appeal to the *HashCollisionResistant* property.

$\langle 5\rangle 1.$ $HLPublicState \in HashDomain$

$\langle 6\rangle 1.$ $HLPublicState \in PublicStateType$

BY $\langle 2\rangle 1$, $\langle 3\rangle 6$ DEF *LL1Refinement*

$\langle 6\rangle 2.$ QED

BY $\langle 6\rangle 1$ DEF *HashDomain*

$\langle 5\rangle 2.$ $\land$ $LL1RAM.publicState \in HashDomain$
$\land$ $LL1RAM.privateStateEnc \in HashDomain$

$\langle 6\rangle 1.$ $\land$ $LL1RAM.publicState \in PublicStateType$
$\land$ $LL1RAM.privateStateEnc \in PrivateStateEncType$

$\langle 7\rangle 1.$ *LL1TypeInvariant*

BY $\langle 2\rangle 1$

$\langle 7\rangle 2.$ QED

BY $\langle 7\rangle 1$, *LL1SubtypeImplicationLemma*DEF *LL1SubtypeImplication*

$\langle 6\rangle 2.$ QED

BY $\langle 6\rangle 1$ DEF *HashDomain*

$\langle 5\rangle 3.$ $refPrivateStateEnc \in HashDomain$

$\langle 6\rangle 1.$ $refPrivateStateEnc \in PrivateStateEncType$

BY $\langle 3\rangle 3$

$\langle 6\rangle 2.$ QED

BY $\langle 6\rangle 1$ DEF *HashDomain*

We then prove that the refined state hash equals the state hash defined by *LL1PerformOperation*. We will use the *UniquenessInvariant*.

$\langle 5\rangle 4.$ *refStateHash* $=$ *stateHash*

We prove some types we will need for the *UniquenessInvariant*.

$\langle 6\rangle 1.$ *refStateHash* $\in$ *HashType*
BY $\langle 3\rangle 3$

$\langle 6\rangle 2.$ *stateHash* $\in$ *HashType*
BY $\langle 3\rangle 5$

We prove that the refined history state binding is authenticated. This follows directly from the *LL1Refinement*.

$\langle 6\rangle 3.$ *LL1HistoryStateBindingAuthenticated*(*refHistoryStateBinding*)
BY $\langle 2\rangle 1$, $\langle 3\rangle 6$ DEF *LL1Refinement*, *refHistoryStateBinding*, *refStateHash*, *refPrivateStateEnc*

We prove that the history state binding defined by the *LL1PerformOperation* action is authenticated.

$\langle 6\rangle 4.$ *LL1HistoryStateBindingAuthenticated*(*historyStateBinding*)

The authenticator in the RAM authenticates the history state binding, since this is an enablement condition of the *LL1PerformOperation* action.

$\langle 7\rangle 1.$ *ValidateMAC*(*LL1NVRAM.symmetricKey*, *historyStateBinding*, *LL1RAM.authenticator*)
BY $\langle 3\rangle 2$ DEF *historyStateBinding*, *stateHash*

From the *UnforgeabilityInvariant*, the authenticator in the RAM is in the set of observed authenticators.

$\langle 7\rangle 2.$ *LL1RAM.authenticator* $\in$ *LL1ObservedAuthenticators*

$\langle 8\rangle 1.$ *UnforgeabilityInvariant*
BY $\langle 2\rangle 1$ DEF *CorrectnessInvariants*

$\langle 8\rangle 2.$ *historyStateBinding* $\in$ *HashType*
BY $\langle 3\rangle 5$

$\langle 8\rangle 3.$ QED
BY $\langle 7\rangle 1$, $\langle 8\rangle 1$, $\langle 8\rangle 2$ DEF *UnforgeabilityInvariant*

The above two conditions satisfy the definition of *LL1HistoryStateBindingAuthenticated*.

$\langle 7\rangle 3.$ QED
BY $\langle 7\rangle 1$, $\langle 7\rangle 2$ DEF *LL1HistoryStateBindingAuthenticated*

Since both history state bindings are authenticated, it follows that the two state hashes are equal.

$\langle 6\rangle 5.$ QED

$\langle 7\rangle 1.$ *UniquenessInvariant*
BY $\langle 2\rangle 1$ DEF *CorrectnessInvariants*

$\langle 7\rangle 2.$ *LL1NVRAM.historySummary* $=$ *LL1RAM.historySummary*
BY $\langle 3\rangle 2$

$\langle 7\rangle 3.$ QED
BY $\langle 7\rangle 1$, $\langle 7\rangle 2$, $\langle 6\rangle 1$, $\langle 6\rangle 2$, $\langle 6\rangle 3$, $\langle 6\rangle 4$
DEF *UniquenessInvariant*, *refHistoryStateBinding*, *historyStateBinding*

$\langle 5\rangle 5.$ QED
BY $\langle 5\rangle 1$, $\langle 5\rangle 2$, $\langle 5\rangle 3$, $\langle 5\rangle 4$, *HashCollisionResistant*DEF *refStateHash*, *stateHash*

From the previous step, it immediately follows that the refined public state matches the public state in the RAM.

$\langle 4\rangle 2.$ *HLPublicState* $=$ *LL1RAM.publicState*
BY $\langle 4\rangle 1$

It is slightly more involved to show that the refined private state matches the private state that is the decryption of the encrypted private state in the RAM. We need to employ the correctness of the symmetric crypto operations.

$\langle 4\rangle 3.$ *HLPrivateState* $=$ *privateState*

$\langle 5\rangle 1.$ *refPrivateStateEnc* $=$ *LL1RAM.privateStateEnc*
BY $\langle 4\rangle 1$

$\langle 5\rangle 2.$ *LL1NVRAM.symmetricKey* $\in$ *SymmetricKeyType*

$\langle 6\rangle 1.$ *LL1TypeInvariant*
BY $\langle 2\rangle 1$

$\langle 6\rangle 2.$ QED

BY $\langle 6\rangle 1$, *LL1SubtypeImplicationLemma* DEF *LL1SubtypeImplication*

$\langle 5\rangle 3$. $HLPrivateState \in PrivateStateType$

BY $\langle 2\rangle 1$, $\langle 3\rangle 6$ DEF *LL1Refinement*

$\langle 5\rangle 4$. QED

BY $\langle 5\rangle 1$, $\langle 5\rangle 2$, $\langle 5\rangle 3$, *SymmetricCryptoCorrect* DEF *refPrivateStateEnc*, *privateState*

$\langle 4\rangle 4$. QED

BY $\langle 4\rangle 2$, $\langle 4\rangle 3$

The second main step is to prove that the public and private states that are produced by the *Service* in *LL1PerformOperation* are identical to the primed values of *HLPublicState* and *HLPrivateState* in the *LL1Refinement*.

$\langle 3\rangle 9$. $\wedge\ HLPublicState' = sResult.newPublicState$
$\wedge\ HLPrivateState' = sResult.newPrivateState$

We show that the primed refined public state matches the public state produced by the invocation of *Service* in *LL1PerformOperation*, and that the primed refined encrypted private state matches the encryption of the private state produced by the invocation of *Service* in *LL1PerformOperation*. We will use the *HashCollisionResistant* property.

$\langle 4\rangle 1$. $\wedge\ HLPublicState' = sResult.newPublicState$
$\wedge\ refPrivateStateEnc' = newPrivateStateEnc$

We prove some types we will need for the *HashCollisionResistant* property.

$\langle 5\rangle 1$. $HLPublicState' \in HashDomain$

$\langle 6\rangle 1$. $HLPublicState' \in PublicStateType$

BY $\langle 2\rangle 1$, $\langle 3\rangle 7$ DEF *LL1Refinement*

$\langle 6\rangle 2$. QED

BY $\langle 6\rangle 1$ DEF *HashDomain*

$\langle 5\rangle 2$. $refPrivateStateEnc' \in HashDomain$

$\langle 6\rangle 1$. $refPrivateStateEnc' \in PrivateStateEncType$

BY $\langle 3\rangle 4$

$\langle 6\rangle 2$. QED

BY $\langle 6\rangle 1$ DEF *HashDomain*

$\langle 5\rangle 3$. $sResult.newPublicState \in HashDomain$

$\langle 6\rangle 1$. $sResult.newPublicState \in PublicStateType$

BY $\langle 3\rangle 5$

$\langle 6\rangle 2$. QED

BY $\langle 6\rangle 1$ DEF *HashDomain*

$\langle 5\rangle 4$. $newPrivateStateEnc \in HashDomain$

$\langle 6\rangle 1$. $newPrivateStateEnc \in PrivateStateEncType$

BY $\langle 3\rangle 5$

$\langle 6\rangle 2$. QED

BY $\langle 6\rangle 1$ DEF *HashDomain*

We then prove that the primed refined state hash equals the new state hash defined by *LL1PerformOperation*. We will use the *UniquenessInvariant*.

$\langle 5\rangle 5$. $refStateHash' = newStateHash$

We prove that the refined history state binding is authenticated in the primed state. This follows directly from the *LL1Refinement*.

$\langle 6\rangle 1$. $LL1HistoryStateBindingAuthenticated(refHistoryStateBinding)'$

BY $\langle 2\rangle 1$, $\langle 3\rangle 7$ DEF *LL1Refinement*, *refHistoryStateBinding*, *refStateHash*, *refPrivateStateEnc*

Ideally, at this point we would prove that the new history state binding defined by the *LL1PerformOperation* action is authenticated in the primed state. However, we cannot prove this, because TLA+ does not allow us to prime an operator, only an expression. If we prime the entire expression, the *newHistoryStateBinding* argument also becomes primed, which changes its meaning. So instead, we merely prove that the new authenticator defined by the *LL1PerformOperation* action authenticates the new history state binding. We will use the *MACComplete* property.

$\langle 6 \rangle 2.$ $ValidateMAC(LL1NVRAM.symmetricKey', newHistoryStateBinding, newAuthenticator)$

The new authenticator was genererated as a *MAC* of the new history state binding by *LL1PerformOperation*, using the unchanged symmetric key in the *NVRAM*.

$\langle 7 \rangle 1.$ $newAuthenticator = GenerateMAC(LL1NVRAM.symmetricKey', newHistoryStateBinding)$

$\langle 8 \rangle 1.$ UNCHANGED $LL1NVRAM.symmetricKey$

BY $\langle 2 \rangle 1$, *SymmetricKeyConstantLemma*

$\langle 8 \rangle 2.$ QED

BY $\langle 3 \rangle 2$, $\langle 8 \rangle 1$

DEF *newAuthenticator*, *newHistoryStateBinding*, *newStateHash*, *newPrivateStateEnc*, *sResult*, *privateState*

We can thus use the *MACComplete* property to show that the generated *MAC* validates appropriately. To do this, we first need to prove some types.

$\langle 7 \rangle 2.$ $LL1NVRAM.symmetricKey' \in SymmetricKeyType$

$\langle 8 \rangle 1.$ $LL1TypeInvariant'$

BY $\langle 2 \rangle 1$

$\langle 8 \rangle 2.$ QED

BY $\langle 8 \rangle 1$, *LL1SubtypeImplicationLemma*DEF *LL1SubtypeImplication*

$\langle 7 \rangle 3.$ $newHistoryStateBinding \in HashType$

BY $\langle 3 \rangle 5$

Then, we appeal to the *MACComplete* property in a straightforward way.

$\langle 7 \rangle 4.$ QED

BY $\langle 7 \rangle 1$, $\langle 7 \rangle 2$, $\langle 7 \rangle 3$, *MACComplete*

Ideally, at this point we would prove that because both history state bindings are authenticated, it follows that the two state hashes are equal. However, we have not proven that the new state binding defined by the *LL1PerformOperation* action is authenticated in the primed state, because TLA+ does not allow us to prime an operator, only an expression. So, we have to expand the definition of *LL1HistoryStateBindingAuthenticated* for the steps of this proof.

$\langle 6 \rangle 3.$ QED

This is the definition of the *UniquenessInvariant*, with the *LETs* instantiated and *LL1HistoryStateBindingAuthenticated* fully expanded.

$$
\begin{aligned}
\langle 7 \rangle 1.\ & \forall\, stateHash1, stateHash2 \in HashType : \\
& \quad (\land\ \exists\, authenticator \in LL1ObservedAuthenticators' : \\
& \qquad\quad ValidateMAC(\\
& \qquad\qquad LL1NVRAM.symmetricKey', \\
& \qquad\qquad Hash(LL1NVRAM.historySummary', stateHash1), \\
& \qquad\qquad authenticator) \\
& \qquad \land\ \exists\, authenticator \in LL1ObservedAuthenticators' : \\
& \qquad\quad ValidateMAC(\\
& \qquad\qquad LL1NVRAM.symmetricKey', \\
& \qquad\qquad Hash(LL1NVRAM.historySummary', stateHash2), \\
& \qquad\qquad authenticator)) \\
& \quad \Rightarrow \\
& \qquad stateHash1 = stateHash2
\end{aligned}
$$

$\langle 8 \rangle 1.$ $UniquenessInvariant'$

BY $\langle 2 \rangle 1$ DEF *CorrectnessInvariants*

$\langle 8 \rangle 2.$ QED

BY $\langle 8 \rangle 1$ DEF *UniquenessInvariant*, *LL1HistoryStateBindingAuthenticated*

The universal quantifiers in the previous step range over *HashType*, sowe need to prove that the state hashes are in *HashType*.

$\langle 7 \rangle 2.$ $refStateHash' \in HashType$

BY $\langle 3 \rangle 4$

$\langle 7 \rangle 3.$ $newStateHash \in HashType$
BY $\langle 3 \rangle 5$

This is the first conjunct in the antecedent of the implication in the expanded *UniquenessInvariant*. It follows directly from the fact that the refined history state binding is authenticated in the primed state.

$\langle 7 \rangle 4.$ $\exists\, authenticator \in LL1ObservedAuthenticators' :$
$ValidateMAC($
$LL1NVRAM.symmetricKey',$
$Hash(LL1NVRAM.historySummary', refStateHash'),$
$authenticator)$
BY $\langle 6 \rangle 1$ DEF *LL1HistoryStateBindingAuthenticated*, *refHistoryStateBinding*

This is the second conjunct in the antecedent of the implication in the expanded *UniquenessInvariant*. The new authenticator defined by the *LL1PerformOperation* action will serve as a witness for the existential quantifier.

$\langle 7 \rangle 5.$ $\exists\, authenticator \in LL1ObservedAuthenticators' :$
$ValidateMAC($
$LL1NVRAM.symmetricKey',$
$Hash(LL1NVRAM.historySummary', newStateHash),$
$authenticator)$

The new authenticator defined by the *LL1PerformOperation* action is in the primed set of observed authenticators, as specified by the *LL1PerformOperation* action.

$\langle 8 \rangle 1.$ $newAuthenticator \in LL1ObservedAuthenticators'$
$\langle 9 \rangle 1.$ $LL1ObservedAuthenticators' =$
$LL1ObservedAuthenticators \cup \{newAuthenticator\}$
BY $\langle 3 \rangle 2$
DEF *newAuthenticator*, *newHistoryStateBinding*, *newStateHash*, *newHistorySummary*, *newPrivateStateEnc*, *sResult*, *privateState*
$\langle 9 \rangle 2.$ QED
BY $\langle 9 \rangle 1$

The new authenticator defined by the *LL1PerformOperation* action authenticates the new history state binding defined by this action. Because the history summary in the primed *NVRAM* equals the new history summary defined by the action, we can derive an expression that satisfies the exsistential expression above.

$\langle 8 \rangle 2.$ $ValidateMAC($
$LL1NVRAM.symmetricKey',$
$Hash(LL1NVRAM.historySummary', newStateHash),$
$newAuthenticator)$
$\langle 9 \rangle 1.$ $newHistoryStateBinding = Hash(LL1NVRAM.historySummary', newStateHash)$
$\langle 10 \rangle 1.$ $LL1NVRAM.historySummary' = newHistorySummary$
$\langle 11 \rangle 1.$ $LL1NVRAM' = [$
$historySummary \mapsto newHistorySummary,$
$symmetricKey \mapsto LL1NVRAM.symmetricKey]$
BY $\langle 3 \rangle 2$ DEF *newHistorySummary*
$\langle 11 \rangle 2.$ QED
BY $\langle 11 \rangle 1$
$\langle 10 \rangle 2.$ QED
BY $\langle 10 \rangle 1$ DEF *newHistoryStateBinding*
$\langle 9 \rangle 2.$ QED
BY $\langle 6 \rangle 2$, $\langle 9 \rangle 1$
$\langle 8 \rangle 3.$ QED
BY $\langle 8 \rangle 1$, $\langle 8 \rangle 2$

With both conjuncts in the antecedent true, the conclusion readily follows.

$\langle 7 \rangle 6.$ QED

BY $\langle 7\rangle 1$, $\langle 7\rangle 2$, $\langle 7\rangle 3$, $\langle 7\rangle 4$, $\langle 7\rangle 5$

$\langle 5\rangle 6$. QED

Ideally, this QED step should just read:

BY $\langle 5\rangle 1$, $\langle 5\rangle 2$, $\langle 5\rangle 3$, $\langle 5\rangle 4$, $\langle 5\rangle 5$, *HashCollisionResistant*

However, the prover seems to get a little confused in this instance. We make life easier for the prover by definining some local variables and hiding their definitions before appealing to the *HashCollisionResistant* assumption.

$\langle 6\rangle$ $h1a \triangleq HLPublicState'$

$\langle 6\rangle$ $h2a \triangleq refPrivateStateEnc'$

$\langle 6\rangle$ $h1b \triangleq sResult.newPublicState$

$\langle 6\rangle$ $h2b \triangleq newPrivateStateEnc$

$\langle 6\rangle 1$. $h1a \in HashDomain$

BY $\langle 5\rangle 1$

$\langle 6\rangle 2$. $h2a \in HashDomain$

BY $\langle 5\rangle 2$

$\langle 6\rangle 3$. $h1b \in HashDomain$

BY $\langle 5\rangle 3$

$\langle 6\rangle 4$. $h2b \in HashDomain$

BY $\langle 5\rangle 4$

$\langle 6\rangle 5$. $Hash(h1a, h2a) = Hash(h1b, h2b)$

BY $\langle 5\rangle 5$ DEF *refStateHash*, *newStateHash*

$\langle 6\rangle 6$. $h1a = h1b \wedge h2a = h2b$

$\langle 7\rangle$ HIDE DEF $h1a$, $h2a$, $h1b$, $h2b$

$\langle 7\rangle 1$. QED

BY $\langle 6\rangle 1$, $\langle 6\rangle 2$, $\langle 6\rangle 3$, $\langle 6\rangle 4$, $\langle 6\rangle 5$, *HashCollisionResistant*

$\langle 6\rangle 7$. QED

BY $\langle 6\rangle 6$

To show that the primed refined high-level private state matches the private state that is produced by the *Service* invocation in *LL1PerformOperation*, we need to appeal to the correctness of the symmetric crypto operations and the *SymmetricKeyConstantLemma*.

$\langle 4\rangle 2$. $HLPrivateState' = sResult.newPrivateState$

$\langle 5\rangle 1$. $SymmetricDecrypt(LL1NVRAM.symmetricKey', refPrivateStateEnc') =$
$SymmetricDecrypt(LL1NVRAM.symmetricKey, newPrivateStateEnc)$

$\langle 6\rangle 1$. $refPrivateStateEnc' = newPrivateStateEnc$

BY $\langle 4\rangle 1$

$\langle 6\rangle 2$. $LL1NVRAM.symmetricKey' = LL1NVRAM.symmetricKey$

BY $\langle 2\rangle 1$, *SymmetricKeyConstantLemma*

$\langle 6\rangle 20$. QED

BY $\langle 6\rangle 1$, $\langle 6\rangle 2$

$\langle 5\rangle 2$. $HLPrivateState' = SymmetricDecrypt(LL1NVRAM.symmetricKey', refPrivateStateEnc')$

$\langle 6\rangle 1$. $LL1NVRAM.symmetricKey' \in SymmetricKeyType$

$\langle 7\rangle 1$. $LL1TypeInvariant'$

BY $\langle 2\rangle 1$

$\langle 7\rangle 2$. QED

BY $\langle 7\rangle 1$, *LL1SubtypeImplicationLemma*DEF *LL1SubtypeImplication*

$\langle 6\rangle 2$. $HLPrivateState' \in PrivateStateType$

BY $\langle 2\rangle 1$, $\langle 3\rangle 7$ DEF *LL1Refinement*

$\langle 6\rangle 3$. QED

BY $\langle 6\rangle 1$, $\langle 6\rangle 2$, *SymmetricCryptoCorrect*DEF *refPrivateStateEnc*

$\langle 5\rangle 3$. $sResult.newPrivateState = SymmetricDecrypt(LL1NVRAM.symmetricKey, newPrivateStateEnc)$

$\langle 6\rangle 1$. $LL1NVRAM.symmetricKey \in SymmetricKeyType$

$\langle 7\rangle 1.$ $LL1\mathit{TypeInvariant}$
BY $\langle 2\rangle 1$
$\langle 7\rangle 2.$ QED
BY $\langle 7\rangle 1$, $LL1\mathit{SubtypeImplicationLemma}$DEF $LL1\mathit{SubtypeImplication}$
$\langle 6\rangle 2.$ $\mathit{sResult.newPrivateState} \in \mathit{PrivateStateType}$
BY $\langle 3\rangle 5$
$\langle 6\rangle 3.$ QED
BY $\langle 6\rangle 1$, $\langle 6\rangle 2$, $\mathit{SymmetricCryptoCorrect}$DEF $\mathit{newPrivateStateEnc}$
$\langle 5\rangle 4.$ QED
BY $\langle 5\rangle 1$, $\langle 5\rangle 2$, $\langle 5\rangle 3$
$\langle 4\rangle 3.$ QED
BY $\langle 4\rangle 1$, $\langle 4\rangle 2$

The QED step states each conjunct of the *HLAdvanceService* action, which all follow directly from the two main steps above.

$\langle 3\rangle 10.$ QED
$\langle 4\rangle 1.$ $\mathit{input} \in \mathit{HLAvailableInputs}$
$\langle 5\rangle 1.$ $\mathit{HLAvailableInputs} = LL1\mathit{AvailableInputs}$
BY $\langle 2\rangle 1$ DEF $LL1\mathit{Refinement}$
$\langle 5\rangle 2.$ QED
BY $\langle 5\rangle 1$, $\langle 3\rangle 2$
$\langle 4\rangle 2.$ $\mathit{HLAlive} = \text{TRUE}$
BY $\langle 2\rangle 1$, $\langle 3\rangle 6$ DEF $LL1\mathit{Refinement}$
$\langle 4\rangle 3.$ $\mathit{HLPublicState}' = \mathit{hlSResult.newPublicState}$
$\langle 5\rangle 1.$ $\mathit{HLPublicState}' = \mathit{sResult.newPublicState}$
BY $\langle 3\rangle 9$
$\langle 5\rangle 2.$ $\mathit{sResult.newPublicState} = \mathit{hlSResult.newPublicState}$
$\langle 6\rangle 1.$ $\mathit{sResult} = \mathit{hlSResult}$
BY $\langle 3\rangle 8$ DEF $\mathit{sResult}$, $\mathit{hlSResult}$
$\langle 6\rangle 2.$ QED
BY $\langle 6\rangle 1$
$\langle 5\rangle 3.$ QED
BY $\langle 5\rangle 1$, $\langle 5\rangle 2$
$\langle 4\rangle 4.$ $\mathit{HLPrivateState}' = \mathit{hlSResult.newPrivateState}$
$\langle 5\rangle 1.$ $\mathit{HLPrivateState}' = \mathit{sResult.newPrivateState}$
BY $\langle 3\rangle 9$
$\langle 5\rangle 2.$ $\mathit{sResult.newPrivateState} = \mathit{hlSResult.newPrivateState}$
$\langle 6\rangle 1.$ $\mathit{sResult} = \mathit{hlSResult}$
BY $\langle 3\rangle 8$ DEF $\mathit{sResult}$, $\mathit{hlSResult}$
$\langle 6\rangle 2.$ QED
BY $\langle 6\rangle 1$
$\langle 5\rangle 3.$ QED
BY $\langle 5\rangle 1$, $\langle 5\rangle 2$
$\langle 4\rangle 5.$ $\mathit{HLObservedOutputs}' = \mathit{HLObservedOutputs} \cup \{\mathit{hlSResult.output}\}$
$\langle 5\rangle 1.$ $LL1\mathit{ObservedOutputs}' = LL1\mathit{ObservedOutputs} \cup \{\mathit{sResult.output}\}$
BY $\langle 3\rangle 2$ DEF $\mathit{sResult}$, $\mathit{privateState}$
$\langle 5\rangle 2.$ $\mathit{HLObservedOutputs} = LL1\mathit{ObservedOutputs}$
BY $\langle 2\rangle 1$ DEF $LL1\mathit{Refinement}$
$\langle 5\rangle 3.$ $\mathit{HLObservedOutputs}' = LL1\mathit{ObservedOutputs}'$
BY $\langle 2\rangle 1$ DEF $LL1\mathit{Refinement}$
$\langle 5\rangle 4.$ $\mathit{sResult.output} = \mathit{hlSResult.output}$
$\langle 6\rangle 1.$ $\mathit{sResult} = \mathit{hlSResult}$
BY $\langle 3\rangle 8$ DEF $\mathit{sResult}$, $\mathit{hlSResult}$

$\langle 6\rangle 2$. QED

BY $\langle 6\rangle 1$

$\langle 5\rangle 5$. QED

BY $\langle 5\rangle 1$, $\langle 5\rangle 2$, $\langle 5\rangle 3$, $\langle 5\rangle 4$

$\langle 4\rangle 6$. UNCHANGED $\mathit{HLAvailableInputs}$

$\langle 5\rangle 1$. UNCHANGED $\mathit{LL1AvailableInputs}$

BY $\langle 3\rangle 2$

$\langle 5\rangle 2$. $\land\ \mathit{HLAvailableInputs} = \mathit{LL1AvailableInputs}$
$\land\ \mathit{HLAvailableInputs}' = \mathit{LL1AvailableInputs}'$

BY $\langle 2\rangle 1$ DEF $\mathit{LL1Refinement}$

$\langle 5\rangle 3$. QED

BY $\langle 5\rangle 1$, $\langle 5\rangle 2$

$\langle 4\rangle 7$. UNCHANGED $\mathit{HLAlive}$

BY $\langle 2\rangle 1$, $\langle 3\rangle 6$, $\langle 3\rangle 7$, $\mathit{LL1NVRAMHistorySummaryUncorruptedEqualsHLAliveLemma}$

$\langle 4\rangle 8$. QED

BY $\langle 4\rangle 1$, $\langle 4\rangle 2$, $\langle 4\rangle 3$, $\langle 4\rangle 4$, $\langle 4\rangle 5$, $\langle 4\rangle 6$, $\langle 4\rangle 7$ DEF $\mathit{HLAdvanceService}$, $\mathit{hlSResult}$

A Memoir-Basic *LL1RepeatOperation* action refines to a high-level stuttering step.

$\langle 2\rangle 5$. $\mathit{LL1RepeatOperation} \Rightarrow$ UNCHANGED HLVars

$\langle 3\rangle 1$. HAVE $\mathit{LL1RepeatOperation}$

$\langle 3\rangle 2$. PICK $\mathit{input} \in \mathit{LL1AvailableInputs} : \mathit{LL1RepeatOperation}!(\mathit{input})!1$

BY $\langle 3\rangle 1$ DEF $\mathit{LL1RepeatOperation}$

The *HLAlive* predicate is unchanged because the *LL1NVRAMHistorySummaryUncorrupted* predicate is unchanged.

$\langle 3\rangle 3$. UNCHANGED $\mathit{HLAlive}$

The *LL1NVRAMHistorySummaryUncorruptedEqualsHLAliveLemma* tells us that *LL1NVRAMHistorySummaryUncorrupted* and *HLAlive* are equal.

$\langle 4\rangle 1$. $\mathit{LL1NVRAMHistorySummaryUncorrupted} = \mathit{HLAlive}$

$\langle 5\rangle 1$. $\mathit{LL1Refinement}$

BY $\langle 2\rangle 1$

$\langle 5\rangle 2$. QED

BY $\langle 5\rangle 1$, $\mathit{LL1NVRAMHistorySummaryUncorruptedEqualsHLAliveLemma}$

$\langle 4\rangle 2$. $\mathit{LL1NVRAMHistorySummaryUncorrupted}' = \mathit{HLAlive}'$

$\langle 5\rangle 1$. $\mathit{LL1Refinement}'$

BY $\langle 2\rangle 1$

$\langle 5\rangle 2$. QED

BY $\langle 5\rangle 1$, $\mathit{LL1NVRAMHistorySummaryUncorruptedEqualsHLAliveLemma}$

Then, the *LL1NVRAMHistorySummaryUncorruptedUnchangedLemma* tells us that the *LL1NVRAMHistorySummaryUncorrupted* predicate is unchanged.

$\langle 4\rangle 3$. UNCHANGED $\mathit{LL1NVRAMHistorySummaryUncorrupted}$

$\langle 5\rangle 1$. $\mathit{LL1TypeInvariant}$

BY $\langle 2\rangle 1$

The *NVRAM* is unchanged by definition of the *LL1RepeatOperation* action.

$\langle 5\rangle 2$. UNCHANGED $\mathit{LL1NVRAM}$

BY $\langle 3\rangle 2$

The *LL1RepeatOperationUnchangedAuthenticatedHistoryStateBindingsLemma* tells us that there is no change to the set of history state bindings that have authenticators in the set *LL1ObservedAuthenticators*.

$\langle 5\rangle 3$. $\forall\ \mathit{historyStateBinding} \in \mathit{HashType} :$
UNCHANGED $\mathit{LL1HistoryStateBindingAuthenticated}(\mathit{historyStateBinding})$

$\langle 6\rangle 1$. $\mathit{LL1TypeInvariant} \land \mathit{UnforgeabilityInvariant} \land \mathit{InclusionInvariant}$

BY $\langle 2\rangle 1$ DEF $\mathit{CorrectnessInvariants}$

⟨6⟩2. QED
BY ⟨3⟩1, ⟨6⟩1, *LL1RepeatOperationUnchangedAuthenticatedHistoryStateBindingsLemma*

⟨5⟩4. QED
BY ⟨5⟩1, ⟨5⟩2, ⟨5⟩3, *LL1NVRAMHistorySummaryUncorruptedUnchangedLemma*

⟨4⟩4. QED
BY ⟨4⟩1, ⟨4⟩2, ⟨4⟩3

The mapping from *LL1AvailableInputs* to *HLAvailableInputs* is direct.

⟨3⟩4. UNCHANGED *HLAvailableInputs*

⟨4⟩1. UNCHANGED *LL1AvailableInputs*
BY ⟨3⟩2

⟨4⟩2. $\wedge$ *HLAvailableInputs* = *LL1AvailableInputs*
$\wedge$ *HLAvailableInputs*′ = *LL1AvailableInputs*′
BY ⟨2⟩1 DEF *LL1Refinement*

⟨4⟩3. QED
BY ⟨4⟩1, ⟨4⟩2

The mapping from *LL1ObservedOutputs* to *HLObservedOutputs* is direct.

⟨3⟩5. UNCHANGED *HLObservedOutputs*

⟨4⟩1. UNCHANGED *LL1ObservedOutputs*

⟨5⟩1. *LL1TypeInvariant* $\wedge$ *UnforgeabilityInvariant* $\wedge$ *InclusionInvariant*
BY ⟨2⟩1 DEF *CorrectnessInvariants*

⟨5⟩2. QED
BY ⟨5⟩1, ⟨3⟩1, *LL1RepeatOperationUnchangedObseredOutputsLemma*

⟨4⟩2. $\wedge$ *HLObservedOutputs* = *LL1ObservedOutputs*
$\wedge$ *HLObservedOutputs*′ = *LL1ObservedOutputs*′
BY ⟨2⟩1 DEF *LL1Refinement*

⟨4⟩3. QED
BY ⟨4⟩1, ⟨4⟩2

We prove the stuttering of the high-level public and private state by using the *NonAdvancementLemma*.

⟨3⟩6. UNCHANGED ⟨*HLPublicState*, *HLPrivateState*⟩

Many of the antecedents for the *NonAdvancementLemma* come directly from antecedents in the induction.

⟨4⟩1. $\wedge$ *LL1Refinement*
$\wedge$ *LL1Refinement*′
$\wedge$ *LL1TypeInvariant*
$\wedge$ *LL1TypeInvariant*′
$\wedge$ *UniquenessInvariant*
BY ⟨2⟩1 DEF *CorrectnessInvariants*

The *NVRAM* is unchanged by definition of the *LL1RepeatOperation* action.

⟨4⟩2. UNCHANGED *LL1NVRAM*
BY ⟨3⟩2

The
LL1RepeatOperationUnchangedAuthenticatedHistoryStateBindingsLemma
tells us that there is no change to the set of history state bindings that have authenticators in the set *LL1ObservedAuthenticators*.

⟨4⟩3. $\forall$ *historyStateBinding* $\in$ *HashType* :
UNCHANGED *LL1HistoryStateBindingAuthenticated*(*historyStateBinding*)

⟨5⟩1. *LL1TypeInvariant* $\wedge$ *UnforgeabilityInvariant* $\wedge$ *InclusionInvariant*
BY ⟨2⟩1 DEF *CorrectnessInvariants*

⟨5⟩2. QED
BY ⟨5⟩1, ⟨3⟩1, *LL1RepeatOperationUnchangedAuthenticatedHistoryStateBindingsLemma*

We have all of the antecedents for the *NonAdvancementLemma*, so we can apply it directly.

$\langle 4 \rangle 4.$ QED

BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$, $\langle 4 \rangle 3$, *NonAdvancementLemma*

$\langle 3 \rangle 7.$ QED

BY $\langle 3 \rangle 3$, $\langle 3 \rangle 4$, $\langle 3 \rangle 5$, $\langle 3 \rangle 6$ DEF *HLVars*

A Memoir-Basic *LL1Restart* action refines to a high-level stuttering step.

$\langle 2 \rangle 6.$ $LL1Restart \Rightarrow$ UNCHANGED *HLVars*

$\langle 3 \rangle 1.$ HAVE *LL1Restart*

$\langle 3 \rangle 2.$ PICK $untrustedStorage \in LL1UntrustedStorageType,$
$randomSymmetricKey \in SymmetricKeyType \setminus \{LL1NVRAM.symmetricKey\},$
$hash \in HashType :$
$LL1Restart!(untrustedStorage, randomSymmetricKey, hash)$

BY $\langle 3 \rangle 1$ DEF *LL1Restart*

The *HLAlive* predicate is unchanged because the *LL1NVRAMHistorySummaryUncorrupted* predicate is unchanged.

$\langle 3 \rangle 3.$ UNCHANGED *HLAlive*

The *LL1NVRAMHistorySummaryUncorruptedEqualsHLAliveLemma* tells us that *LL1NVRAMHistorySummaryUncorrupted* and *HLAlive* are equal.

$\langle 4 \rangle 1.$ $LL1NVRAMHistorySummaryUncorrupted = HLAlive$

$\langle 5 \rangle 1.$ *LL1Refinement*

BY $\langle 2 \rangle 1$

$\langle 5 \rangle 2.$ QED

BY $\langle 5 \rangle 1$, *LL1NVRAMHistorySummaryUncorruptedEqualsHLAliveLemma*

$\langle 4 \rangle 2.$ $LL1NVRAMHistorySummaryUncorrupted' = HLAlive'$

$\langle 5 \rangle 1.$ $LL1Refinement'$

BY $\langle 2 \rangle 1$

$\langle 5 \rangle 2.$ QED

BY $\langle 5 \rangle 1$, *LL1NVRAMHistorySummaryUncorruptedEqualsHLAliveLemma*

Then, the *LL1NVRAMHistorySummaryUncorruptedUnchangedLemma* tells us that the *LL1NVRAMHistorySummaryUncorrupted* predicate is unchanged.

$\langle 4 \rangle 3.$ UNCHANGED *LL1NVRAMHistorySummaryUncorrupted*

$\langle 5 \rangle 1.$ *LL1TypeInvariant*

BY $\langle 2 \rangle 1$

The *NVRAM* is unchanged by definition of the *LL1Restart* action.

$\langle 5 \rangle 2.$ UNCHANGED *LL1NVRAM*

BY $\langle 3 \rangle 2$

The *UnchangedAuthenticatedHistoryStateBindingsLemma* tells us that there is no change to the set of history state bindings that have authenticators in the set *LL1ObservedAuthenticators*.

$\langle 5 \rangle 3.$ $\forall historyStateBinding \in HashType :$
UNCHANGED $LL1HistoryStateBindingAuthenticated(historyStateBinding)$

$\langle 6 \rangle 1.$ UNCHANGED $\langle LL1NVRAM, LL1ObservedAuthenticators \rangle$

BY $\langle 3 \rangle 2$

$\langle 6 \rangle 2.$ QED

BY $\langle 6 \rangle 1$, *UnchangedAuthenticatedHistoryStateBindingsLemma*

We have all of the antecedents for the *LL1NVRAMHistorySummaryUncorruptedUnchangedLemma*, so we can apply it directly.

$\langle 5 \rangle 4.$ QED

BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$, $\langle 5 \rangle 3$, *LL1NVRAMHistorySummaryUncorruptedUnchangedLemma*

$\langle 4 \rangle 4.$ QED

BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$, $\langle 4 \rangle 3$

The mapping from *LL1AvailableInputs* to *HLAvailableInputs* is direct.

$\langle 3 \rangle 4.$ UNCHANGED *HLAvailableInputs*

$\langle 4 \rangle 1.$ UNCHANGED *LL1AvailableInputs*

BY ⟨3⟩2

⟨4⟩2. $\land$ *HLAvailableInputs* $=$ *LL1AvailableInputs*
$\land$ *HLAvailableInputs*$'$ $=$ *LL1AvailableInputs*$'$

BY ⟨2⟩1 DEF *LL1Refinement*

⟨4⟩3. QED

BY ⟨4⟩1, ⟨4⟩2

The mapping from *LL1ObservedOutputs* to *HLObservedOutputs* is direct.

⟨3⟩5. UNCHANGED *HLObservedOutputs*

⟨4⟩1. UNCHANGED *LL1ObservedOutputs*

BY ⟨3⟩2

⟨4⟩2. $\land$ *HLObservedOutputs* $=$ *LL1ObservedOutputs*
$\land$ *HLObservedOutputs*$'$ $=$ *LL1ObservedOutputs*$'$

BY ⟨2⟩1 DEF *LL1Refinement*

⟨4⟩3. QED

BY ⟨4⟩1, ⟨4⟩2

We prove the stuttering of the high-level public and private state by using the *NonAdvancementLemma*.

⟨3⟩6. UNCHANGED ⟨*HLPublicState*, *HLPrivateState*⟩

Many of the antecedents for the *NonAdvancementLemma* come directly from antecedents in the induction.

⟨4⟩1. $\land$ *LL1Refinement*
$\land$ *LL1Refinement*$'$
$\land$ *LL1TypeInvariant*
$\land$ *LL1TypeInvariant*$'$
$\land$ *UniquenessInvariant*

BY ⟨2⟩1 DEF *CorrectnessInvariants*

The *LL1NVRAM* is unchanged by definition of the *LL1Restart* action.

⟨4⟩2. UNCHANGED *LL1NVRAM*

BY ⟨3⟩2

The *UnchangedAuthenticatedHistoryStateBindingsLemma* tells us that there is no change to the set of history state bindings that have authenticators in the set *LL1ObservedAuthenticators*.

⟨4⟩3. $\forall$ *historyStateBinding* $\in$ *HashType* :
UNCHANGED *LL1HistoryStateBindingAuthenticated*(*historyStateBinding*)

⟨5⟩1. UNCHANGED ⟨*LL1NVRAM*, *LL1ObservedAuthenticators*⟩

BY ⟨3⟩2

⟨5⟩2. QED

BY ⟨5⟩1, *UnchangedAuthenticatedHistoryStateBindingsLemma*

We have all of the antecedents for the *NonAdvancementLemma*, so we can apply it directly.

⟨4⟩4. QED

BY ⟨4⟩1, ⟨4⟩2, ⟨4⟩3, *NonAdvancementLemma*

⟨3⟩7. QED

BY ⟨3⟩3, ⟨3⟩4, ⟨3⟩5, ⟨3⟩6 DEF *HLVars*

A Memoir-Basic *LL1ReadDisk* action refines to a high-level stuttering step.

⟨2⟩7. *LL1ReadDisk* $\Rightarrow$ UNCHANGED *HLVars*

⟨3⟩1. HAVE *LL1ReadDisk*

The *HLAlive* predicate is unchanged because the *LL1NVRAMHistorySummaryUncorrupted* predicate is unchanged.

⟨3⟩2. UNCHANGED *HLAlive*

The *LL1NVRAMHistorySummaryUncorruptedEqualsHLAliveLemma* tells us that *LL1NVRAMHistorySummaryUncorrupted* and *HLAlive* are equal.

⟨4⟩1. *LL1NVRAMHistorySummaryUncorrupted* $=$ *HLAlive*

⟨5⟩1. *LL1Refinement*

BY ⟨2⟩1

⟨5⟩2. QED
BY ⟨5⟩1, *LL1NVRAMHistorySummaryUncorruptedEqualsHLAliveLemma*

⟨4⟩2. $LL1NVRAMHistorySummaryUncorrupted' = HLAlive'$

⟨5⟩1. $LL1Refinement'$
BY ⟨2⟩1

⟨5⟩2. QED
BY ⟨5⟩1, *LL1NVRAMHistorySummaryUncorruptedEqualsHLAliveLemma*

Then, the *LL1NVRAMHistorySummaryUncorruptedUnchangedLemma* tells us that the *LL1NVRAMHistorySummaryUncorrupted* predicate is unchanged.

⟨4⟩3. UNCHANGED *LL1NVRAMHistorySummaryUncorrupted*

⟨5⟩1. *LL1TypeInvariant*
BY ⟨2⟩1

The *NVRAM* is unchanged by definition of the *LL1ReadDisk* action.

⟨5⟩2. UNCHANGED *LL1NVRAM*
BY ⟨3⟩1 DEF *LL1ReadDisk*

The *UnchangedAuthenticatedHistoryStateBindingsLemma* tells us that there is no change to the set of history state bindings that have authenticators in the set *LL1ObservedAuthenticators*.

⟨5⟩3. $\forall$ *historyStateBinding* $\in$ *HashType* :
UNCHANGED *LL1HistoryStateBindingAuthenticated*(*historyStateBinding*)

⟨6⟩1. UNCHANGED ⟨*LL1NVRAM*, *LL1ObservedAuthenticators*⟩
BY ⟨3⟩1 DEF *LL1ReadDisk*

⟨6⟩2. QED
BY ⟨6⟩1, *UnchangedAuthenticatedHistoryStateBindingsLemma*

We have all of the antecedents for the *LL1NVRAMHistorySummaryUncorruptedUnchangedLemma*, so we can apply it directly.

⟨5⟩4. QED
BY ⟨5⟩1, ⟨5⟩2, ⟨5⟩3, *LL1NVRAMHistorySummaryUncorruptedUnchangedLemma*

⟨4⟩4. QED
BY ⟨4⟩1, ⟨4⟩2, ⟨4⟩3

The mapping from *LL1AvailableInputs* to *HLAvailableInputs* is direct.

⟨3⟩3. UNCHANGED *HLAvailableInputs*

⟨4⟩1. UNCHANGED *LL1AvailableInputs*
BY ⟨3⟩1 DEF *LL1ReadDisk*

⟨4⟩2. $\land\ HLAvailableInputs = LL1AvailableInputs$
$\land\ HLAvailableInputs' = LL1AvailableInputs'$
BY ⟨2⟩1 DEF *LL1Refinement*

⟨4⟩3. QED
BY ⟨4⟩1, ⟨4⟩2

The mapping from *LL1ObservedOutputs* to *HLObservedOutputs* is direct.

⟨3⟩4. UNCHANGED *HLObservedOutputs*

⟨4⟩1. UNCHANGED *LL1ObservedOutputs*
BY ⟨3⟩1 DEF *LL1ReadDisk*

⟨4⟩2. $\land\ HLObservedOutputs = LL1ObservedOutputs$
$\land\ HLObservedOutputs' = LL1ObservedOutputs'$
BY ⟨2⟩1 DEF *LL1Refinement*

⟨4⟩3. QED
BY ⟨4⟩1, ⟨4⟩2

We prove the stuttering of the high-level public and private state by using the *NonAdvancementLemma*.

⟨3⟩5. UNCHANGED ⟨*HLPublicState*, *HLPrivateState*⟩

Many of the antecedents for the *NonAdvancementLemma* come directly from antecedents in the induction.

$\langle 4\rangle 1.\ \wedge\ \mathit{LL1Refinement}$
$\quad\wedge\ \mathit{LL1Refinement}'$
$\quad\wedge\ \mathit{LL1TypeInvariant}$
$\quad\wedge\ \mathit{LL1TypeInvariant}'$
$\quad\wedge\ \mathit{UniquenessInvariant}$
BY $\langle 2\rangle 1$ DEF *CorrectnessInvariants*

The *NVRAM* is unchanged by definition of the *LL1ReadDisk* action.

$\langle 4\rangle 2.$ UNCHANGED *LL1NVRAM*
BY $\langle 3\rangle 1$ DEF *LL1ReadDisk*

The *UnchangedAuthenticatedHistoryStateBindingsLemma* tells us that there is no change to the set of history state bindings that have authenticators in the set *LL1ObservedAuthenticators*.

$\langle 4\rangle 3.\ \forall\ \mathit{historyStateBinding} \in \mathit{HashType}:$
UNCHANGED $\mathit{LL1HistoryStateBindingAuthenticated}(\mathit{historyStateBinding})$
$\langle 5\rangle 1.$ UNCHANGED $\langle \mathit{LL1NVRAM}, \mathit{LL1ObservedAuthenticators}\rangle$
BY $\langle 3\rangle 1$ DEF *LL1ReadDisk*
$\langle 5\rangle 2.$ QED
BY $\langle 5\rangle 1$, *UnchangedAuthenticatedHistoryStateBindingsLemma*

We have all of the antecedents for the *NonAdvancementLemma*, so we can apply it directly.

$\langle 4\rangle 4.$ QED
BY $\langle 4\rangle 1$, $\langle 4\rangle 2$, $\langle 4\rangle 3$, *NonAdvancementLemma*
$\langle 3\rangle 6.$ QED
BY $\langle 3\rangle 2$, $\langle 3\rangle 3$, $\langle 3\rangle 4$, $\langle 3\rangle 5$ DEF *HLVars*

A Memoir-Basic *LL1WriteDisk* action refines to a high-level stuttering step.

$\langle 2\rangle 8.\ \mathit{LL1WriteDisk} \Rightarrow$ UNCHANGED *HLVars*
$\langle 3\rangle 1.$ HAVE *LL1WriteDisk*

The *HLAlive* predicate is unchanged because the *LL1NVRAMHistorySummaryUncorrupted* predicate is unchanged.

$\langle 3\rangle 2.$ UNCHANGED *HLAlive*

The *LL1NVRAMHistorySummaryUncorruptedEqualsHLAliveLemma* tells us that *LL1NVRAMHistorySummaryUncorrupted* and *HLAlive* are equal.

$\langle 4\rangle 1.\ \mathit{LL1NVRAMHistorySummaryUncorrupted} = \mathit{HLAlive}$
$\langle 5\rangle 1.$ *LL1Refinement*
BY $\langle 2\rangle 1$
$\langle 5\rangle 2.$ QED
BY $\langle 5\rangle 1$, *LL1NVRAMHistorySummaryUncorruptedEqualsHLAliveLemma*
$\langle 4\rangle 2.\ \mathit{LL1NVRAMHistorySummaryUncorrupted}' = \mathit{HLAlive}'$
$\langle 5\rangle 1.\ \mathit{LL1Refinement}'$
BY $\langle 2\rangle 1$
$\langle 5\rangle 2.$ QED
BY $\langle 5\rangle 1$, *LL1NVRAMHistorySummaryUncorruptedEqualsHLAliveLemma*

Then, the *LL1NVRAMHistorySummaryUncorruptedUnchangedLemma* tells us that the *LL1NVRAMHistorySummaryUncorrupted* predicate is unchanged.

$\langle 4\rangle 3.$ UNCHANGED *LL1NVRAMHistorySummaryUncorrupted*
$\langle 5\rangle 1.$ *LL1TypeInvariant*
BY $\langle 2\rangle 1$

The *NVRAM* is unchanged by definition of the *LL1WriteDisk* action.

$\langle 5\rangle 2.$ UNCHANGED *LL1NVRAM*
BY $\langle 3\rangle 1$ DEF *LL1WriteDisk*

The *UnchangedAuthenticatedHistoryStateBindingsLemma* tells us that there is no change to the set of history state bindings that have authenticators in the set *LL1ObservedAuthenticators*.

$\langle 5\rangle 3.\ \forall\ \mathit{historyStateBinding} \in \mathit{HashType}:$

UNCHANGED $LL1HistoryStateBindingAuthenticated(historyStateBinding)$

$\langle 6\rangle 1.$ UNCHANGED $\langle LL1NVRAM, LL1ObservedAuthenticators\rangle$

BY $\langle 3\rangle 1$ DEF $LL1WriteDisk$

$\langle 6\rangle 2.$ QED

BY $\langle 6\rangle 1$, $UnchangedAuthenticatedHistoryStateBindingsLemma$

We have all of the antecedents for the *LL1NVRAMHistorySummaryUncorruptedUnchangedLemma*, so we can apply it directly.

$\langle 5\rangle 4.$ QED

BY $\langle 5\rangle 1$, $\langle 5\rangle 2$, $\langle 5\rangle 3$, $LL1NVRAMHistorySummaryUncorruptedUnchangedLemma$

$\langle 4\rangle 4.$ QED

BY $\langle 4\rangle 1$, $\langle 4\rangle 2$, $\langle 4\rangle 3$

The mapping from *LL1AvailableInputs* to *HLAvailableInputs* is direct.

$\langle 3\rangle 3.$ UNCHANGED $HLAvailableInputs$

$\langle 4\rangle 1.$ UNCHANGED $LL1AvailableInputs$

BY $\langle 3\rangle 1$ DEF $LL1WriteDisk$

$\langle 4\rangle 2.$ $\wedge\ HLAvailableInputs = LL1AvailableInputs$

$\wedge\ HLAvailableInputs' = LL1AvailableInputs'$

BY $\langle 2\rangle 1$ DEF $LL1Refinement$

$\langle 4\rangle 3.$ QED

BY $\langle 4\rangle 1$, $\langle 4\rangle 2$

The mapping from *LL1ObservedOutputs* to *HLObservedOutputs* is direct.

$\langle 3\rangle 4.$ UNCHANGED $HLObservedOutputs$

$\langle 4\rangle 1.$ UNCHANGED $LL1ObservedOutputs$

BY $\langle 3\rangle 1$ DEF $LL1WriteDisk$

$\langle 4\rangle 2.$ $\wedge\ HLObservedOutputs = LL1ObservedOutputs$

$\wedge\ HLObservedOutputs' = LL1ObservedOutputs'$

BY $\langle 2\rangle 1$ DEF $LL1Refinement$

$\langle 4\rangle 3.$ QED

BY $\langle 4\rangle 1$, $\langle 4\rangle 2$

We prove the stuttering of the high-level public and private state by using the *NonAdvancementLemma*.

$\langle 3\rangle 5.$ UNCHANGED $\langle HLPublicState, HLPrivateState\rangle$

Many of the antecedents for the *NonAdvancementLemma* come directly from antecedents in the induction.

$\langle 4\rangle 1.$ $\wedge\ LL1Refinement$

$\wedge\ LL1Refinement'$

$\wedge\ LL1TypeInvariant$

$\wedge\ LL1TypeInvariant'$

$\wedge\ UniquenessInvariant$

BY $\langle 2\rangle 1$ DEF $CorrectnessInvariants$

The *NVRAM* is unchanged by definition of the *LL1WriteDisk* action.

$\langle 4\rangle 2.$ UNCHANGED $LL1NVRAM$

BY $\langle 3\rangle 1$ DEF $LL1WriteDisk$

The *UnchangedAuthenticatedHistoryStateBindingsLemma* tells us that there is no change to the set of history state bindings that have authenticators in the set *LL1ObservedAuthenticators*.

$\langle 4\rangle 3.$ $\forall\, historyStateBinding \in HashType :$

UNCHANGED $LL1HistoryStateBindingAuthenticated(historyStateBinding)$

$\langle 5\rangle 1.$ UNCHANGED $\langle LL1NVRAM, LL1ObservedAuthenticators\rangle$

BY $\langle 3\rangle 1$ DEF $LL1WriteDisk$

$\langle 5\rangle 2.$ QED

BY $\langle 5\rangle 1$, $UnchangedAuthenticatedHistoryStateBindingsLemma$

We have all of the antecedents for the *NonAdvancementLemma*, so we can apply it directly.

⟨4⟩4. QED

BY ⟨4⟩1, ⟨4⟩2, ⟨4⟩3, *NonAdvancementLemma*

⟨3⟩6. QED

BY ⟨3⟩2, ⟨3⟩3, ⟨3⟩4, ⟨3⟩5 DEF *HLVars*

A Memoir-Basic *LL1CorruptRAM* action refines to a high-level stuttering step.

⟨2⟩9. $LL1CorruptRAM \Rightarrow$ UNCHANGED *HLVars*

⟨3⟩1. HAVE *LL1CorruptRAM*

⟨3⟩2. PICK $untrustedStorage \in LL1UntrustedStorageType$,
$fakeSymmetricKey \in SymmetricKeyType \setminus \{LL1NVRAM.symmetricKey\}$,
$hash \in HashType$:
$LL1CorruptRAM!(untrustedStorage, fakeSymmetricKey, hash)$

BY ⟨3⟩1 DEF *LL1CorruptRAM*

The *HLAlive* predicate is unchanged because the *LL1NVRAMHistorySummaryUncorrupted* predicate is unchanged.

⟨3⟩3. UNCHANGED *HLAlive*

The *LL1NVRAMHistorySummaryUncorruptedEqualsHLAliveLemma* tells us that *LL1NVRAMHistorySummaryUncorrupted* and *HLAlive* are equal.

⟨4⟩1. $LL1NVRAMHistorySummaryUncorrupted = HLAlive$

⟨5⟩1. *LL1Refinement*

BY ⟨2⟩1

⟨5⟩2. QED

BY ⟨5⟩1, *LL1NVRAMHistorySummaryUncorruptedEqualsHLAliveLemma*

⟨4⟩2. $LL1NVRAMHistorySummaryUncorrupted' = HLAlive'$

⟨5⟩1. $LL1Refinement'$

BY ⟨2⟩1

⟨5⟩2. QED

BY ⟨5⟩1, *LL1NVRAMHistorySummaryUncorruptedEqualsHLAliveLemma*

Then, the *LL1NVRAMHistorySummaryUncorruptedUnchangedLemma* tells us that the *LL1NVRAMHistorySummaryUncorrupted* predicate is unchanged.

⟨4⟩3. UNCHANGED *LL1NVRAMHistorySummaryUncorrupted*

⟨5⟩1. *LL1TypeInvariant*

BY ⟨2⟩1

The *LL1NVRAM* is unchanged by definition of the *LL1CorruptRAM* action.

⟨5⟩2. UNCHANGED *LL1NVRAM*

BY ⟨3⟩2

The *UnchangedAuthenticatedHistoryStateBindingsLemma* tells us that there is no change to the set of history state bindings that have authenticators in the set *LL1ObservedAuthenticators*.

⟨5⟩3. $\forall historyStateBinding \in HashType$:
UNCHANGED $LL1HistoryStateBindingAuthenticated(historyStateBinding)$

⟨6⟩1. UNCHANGED $\langle LL1NVRAM, LL1ObservedAuthenticators \rangle$

BY ⟨3⟩2

⟨6⟩2. QED

BY ⟨6⟩1, *UnchangedAuthenticatedHistoryStateBindingsLemma*

We have all of the antecedents for the *LL1NVRAMHistorySummaryUncorruptedUnchangedLemma*, so we can apply it directly.

⟨5⟩4. QED

BY ⟨5⟩1, ⟨5⟩2, ⟨5⟩3, *LL1NVRAMHistorySummaryUncorruptedUnchangedLemma*

⟨4⟩4. QED

BY ⟨4⟩1, ⟨4⟩2, ⟨4⟩3

The mapping from *LL1AvailableInputs* to *HLAvailableInputs* is direct.

⟨3⟩4. UNCHANGED *HLAvailableInputs*

⟨4⟩1. UNCHANGED *LL1AvailableInputs*

BY ⟨3⟩2

⟨4⟩2. $\land$ *HLAvailableInputs* = *LL1AvailableInputs*
$\land$ *HLAvailableInputs*′ = *LL1AvailableInputs*′

BY ⟨2⟩1 DEF *LL1Refinement*

⟨4⟩3. QED

BY ⟨4⟩1, ⟨4⟩2

The mapping from *LL1ObservedOutputs* to *HLObservedOutputs* is direct.

⟨3⟩5. UNCHANGED *HLObservedOutputs*

⟨4⟩1. UNCHANGED *LL1ObservedOutputs*

BY ⟨3⟩2

⟨4⟩2. $\land$ *HLObservedOutputs* = *LL1ObservedOutputs*
$\land$ *HLObservedOutputs*′ = *LL1ObservedOutputs*′

BY ⟨2⟩1 DEF *LL1Refinement*

⟨4⟩3. QED

BY ⟨4⟩1, ⟨4⟩2

We prove the stuttering of the high-level public and private state by using the *NonAdvancementLemma*.

⟨3⟩6. UNCHANGED ⟨*HLPublicState*, *HLPrivateState*⟩

Many of the antecedents for the *NonAdvancementLemma* come directly from antecedents in the induction.

⟨4⟩1. $\land$ *LL1Refinement*
$\land$ *LL1Refinement*′
$\land$ *LL1TypeInvariant*
$\land$ *LL1TypeInvariant*′
$\land$ *UniquenessInvariant*

BY ⟨2⟩1 DEF *CorrectnessInvariants*

The *NVRAM* is unchanged by definition of the *LL1CorruptRAM* action.

⟨4⟩2. UNCHANGED *LL1NVRAM*

BY ⟨3⟩2

The *UnchangedAuthenticatedHistoryStateBindingsLemma* tells us that there is no change to the set of history state bindings that have authenticators in the set *LL1ObservedAuthenticators*.

⟨4⟩3. $\forall$ *historyStateBinding* $\in$ *HashType* :
UNCHANGED *LL1HistoryStateBindingAuthenticated*(*historyStateBinding*)

⟨5⟩1. UNCHANGED ⟨*LL1NVRAM*, *LL1ObservedAuthenticators*⟩

BY ⟨3⟩2

⟨5⟩2. QED

BY ⟨5⟩1, *UnchangedAuthenticatedHistoryStateBindingsLemma*

We have all of the antecedents for the *NonAdvancementLemma*, so we can apply it directly.

⟨4⟩4. QED

BY ⟨4⟩1, ⟨4⟩2, ⟨4⟩3, *NonAdvancementLemma*

⟨3⟩7. QED

BY ⟨3⟩3, ⟨3⟩4, ⟨3⟩5, ⟨3⟩6 DEF *HLVars*

A Memoir-Basic *LL1RestrictedCorruption* action refines to a high-level *HLDie* step.

⟨2⟩10. *LL1RestrictedCorruption* $\Rightarrow$ *HLDie*

⟨3⟩1. HAVE *LL1RestrictedCorruption*

⟨3⟩2. PICK *garbageHistorySummary* $\in$ *HashType* :
LL1RestrictedCorruption!*nvram*!(*garbageHistorySummary*)

BY ⟨3⟩1 DEF *LL1RestrictedCorruption*

First, we prove that this action causes the *LL1NVRAMHistorySummaryUncorrupted* predicate to become false.

⟨3⟩3. *LL1NVRAMHistorySummaryUncorrupted*′ = FALSE

We will make use of the conjunct in *LL1RestrictedCorruption* that prevents the garbage history summary from being in an authenticated history state binding. This conjunct states a 2-way universally quantifiied predicate.

$\langle 4 \rangle 1.$ $LL1RestrictedCorruption\,!\,nvram\,!\,current(garbageHistorySummary)$
BY $\langle 3 \rangle 2$

The following equivalence, plus the knowledge that the symmetric key in the $NVRAM$ and the set of observed authenticators have not changed, are sufficient to prove that the above 2-way universally quantified predicate equals the negation of the 2-way universally quantified predicate in $LL1NVRAMHistorySummaryUncorrupted$, when expanded through $LL1HistoryStateBindingAuthenticated$.

$\langle 4 \rangle 2.$ $LL1NVRAM.historySummary' = garbageHistorySummary$
$\langle 5 \rangle 1.$ $LL1NVRAM' = [historySummary \mapsto garbageHistorySummary, symmetricKey \mapsto LL1NVRAM.symmetricKey]$
BY $\langle 3 \rangle 2$
$\langle 5 \rangle 2.$ QED
BY $\langle 5 \rangle 1$
$\langle 4 \rangle 3.$ UNCHANGED $LL1NVRAM.symmetricKey$
BY $\langle 2 \rangle 1$, $SymmetricKeyConstantLemma$
$\langle 4 \rangle 4.$ UNCHANGED $LL1ObservedAuthenticators$
BY $\langle 3 \rangle 1$ DEF $LL1RestrictedCorruption$
$\langle 4 \rangle 5.$ QED
BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$, $\langle 4 \rangle 3$, $\langle 4 \rangle 4$
DEF $LL1NVRAMHistorySummaryUncorrupted$, $LL1HistoryStateBindingAuthenticated$

Because the $LL1NVRAMHistorySummaryUncorrupted$ is false, the $LL1Refinement$ immediately tells us that the high-level system is not alive.

$\langle 3 \rangle 4.$ $HLAlive' =$ FALSE
BY $\langle 2 \rangle 1$, $\langle 3 \rangle 3$ DEF $LL1Refinement$

The mapping from $LL1AvailableInputs$ to $HLAvailableInputs$ is direct.

$\langle 3 \rangle 5.$ UNCHANGED $HLAvailableInputs$
$\langle 4 \rangle 1.$ UNCHANGED $LL1AvailableInputs$
BY $\langle 3 \rangle 1$ DEF $LL1RestrictedCorruption$
$\langle 4 \rangle 2.$ $\land\ HLAvailableInputs = LL1AvailableInputs$
$\land\ HLAvailableInputs' = LL1AvailableInputs'$
BY $\langle 2 \rangle 1$ DEF $LL1Refinement$
$\langle 4 \rangle 3.$ QED
BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$

The mapping from $LL1ObservedOutputs$ to $HLObservedOutputs$ is direct.

$\langle 3 \rangle 6.$ UNCHANGED $HLObservedOutputs$
$\langle 4 \rangle 1.$ UNCHANGED $LL1ObservedOutputs$
BY $\langle 3 \rangle 1$ DEF $LL1RestrictedCorruption$
$\langle 4 \rangle 2.$ $\land\ HLObservedOutputs = LL1ObservedOutputs$
$\land\ HLObservedOutputs' = LL1ObservedOutputs'$
BY $\langle 2 \rangle 1$ DEF $LL1Refinement$
$\langle 4 \rangle 3.$ QED
BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$

Because the $LL1NVRAMHistorySummaryUncorrupted$ is false, the $LL1Refinement$ immediately tells us that the high-level public state equals the dead state.

$\langle 3 \rangle 7.$ $HLPublicState' = DeadPublicState$
BY $\langle 2 \rangle 1$, $\langle 3 \rangle 3$ DEF $LL1Refinement$

Because the $LL1NVRAMHistorySummaryUncorrupted$ is false, the $LL1Refinement$ immediately tells us that the high-level private state equals the dead state.

$\langle 3 \rangle 8.$ $HLPrivateState' = DeadPrivateState$
BY $\langle 2 \rangle 1$, $\langle 3 \rangle 3$ DEF $LL1Refinement$
$\langle 3 \rangle 9.$ QED
BY $\langle 3 \rangle 4$, $\langle 3 \rangle 5$, $\langle 3 \rangle 6$, $\langle 3 \rangle 7$, $\langle 3 \rangle 8$ DEF $HLDie$

$\langle 2 \rangle 11.$ QED

BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, $\langle 2 \rangle 3$, $\langle 2 \rangle 4$, $\langle 2 \rangle 5$, $\langle 2 \rangle 6$, $\langle 2 \rangle 7$, $\langle 2 \rangle 8$, $\langle 2 \rangle 9$, $\langle 2 \rangle 10$ DEF $HLNext$, $LL1Next$

$\langle 1 \rangle 5.$ QED

Using the *StepSimulation* proof rule, the base case and the induction step together imply that the invariant always holds.

$\langle 2 \rangle 1.$ $\Box[LL1Next]_{LL1Vars} \land \Box LL1Refinement \land \Box CorrectnessInvariants \Rightarrow \Box[HLNext]_{HLVars}$

BY $\langle 1 \rangle 4$, *StepSimulation*

$\langle 2 \rangle 2.$ QED

BY $\langle 1 \rangle 1$, $\langle 1 \rangle 2$, $\langle 1 \rangle 3$, $\langle 2 \rangle 1$ DEF $LL1Spec$, $HLSpec$, $LL1Refinement$

4.8 Proofs of Lemmas Relating to Types in the Memoir-Opt Spec

MODULE *MemoirLL2TypeLemmas*

This module states and proves sevaral lemmas that are useful for proving type safety. Since type safety is an important part of the implementation proof, these lemmas also will be used in theorems other than the Memoir-Opt type-safety theorem.

The lemmas in this module are:
CheckpointDefsTypeSafeLemma
CheckpointTypeSafe
SuccessorDefsTypeSafeLemma
SuccessorTypeSafe
LL2SubtypeImplicationLemma
LL2InitDefsTypeSafeLemma
LL2PerformOperationDefsTypeSafeLemma
LL2RepeatOperationDefsTypeSafeLemma
LL2TakeCheckpointDefsTypeSafeLemma
LL2CorruptSPCRDefsTypeSafeLemma
AuthenticatorsMatchDefsTypeSafeLemma

EXTENDS *MemoirLL2Refinement*

The *CheckpointDefsTypeSafeLemma* proves that the definitions within the LET of the *Checkpoint* function all have the appropriate type. This is a trivial proof that merely walks through the definitions.

THEOREM *CheckpointDefsTypeSafeLemma* $\triangleq$
 $\forall\, historySummary \in HistorySummaryType$:
  LET
   $checkpointedAnchor \triangleq Hash(historySummary.anchor,\ historySummary.extension)$
   $checkpointedHistorySummary \triangleq [$
    $anchor \mapsto checkpointedAnchor,$
    $extension \mapsto BaseHashValue]$
  IN
   $\land\ checkpointedAnchor \in HashType$
   $\land\ checkpointedHistorySummary \in HistorySummaryType$
   $\land\ checkpointedHistorySummary.anchor \in HashType$
   $\land\ checkpointedHistorySummary.extension \in HashType$
$\langle 1 \rangle 1.$ TAKE $historySummary \in HistorySummaryType$
$\langle 1 \rangle$ $checkpointedAnchor \triangleq Hash(historySummary.anchor,\ historySummary.extension)$
$\langle 1 \rangle$ $checkpointedHistorySummary \triangleq [$
  $anchor \mapsto checkpointedAnchor,$
  $extension \mapsto BaseHashValue]$
$\langle 1 \rangle$ HIDE DEF $checkpointedAnchor,\ checkpointedHistorySummary$
$\langle 1 \rangle 2.$ $checkpointedAnchor \in HashType$
 $\langle 2 \rangle 1.$ $historySummary.anchor \in HashDomain$
  $\langle 3 \rangle 1.$ $historySummary.anchor \in HashType$
   BY $\langle 1 \rangle 1$ DEF $HistorySummaryType$
  $\langle 3 \rangle 2.$ QED
   BY $\langle 3 \rangle 1$ DEF $HashDomain$
 $\langle 2 \rangle 2.$ $historySummary.extension \in HashDomain$
  $\langle 3 \rangle 1.$ $historySummary.extension \in HashType$
   BY $\langle 1 \rangle 1$ DEF $HistorySummaryType$
  $\langle 3 \rangle 2.$ QED
   BY $\langle 3 \rangle 1$ DEF $HashDomain$
 $\langle 2 \rangle 3.$ QED

BY $\langle 2\rangle 1$, $\langle 2\rangle 2$, *HashTypeSafe*DEF *checkpointedAnchor*

$\langle 1\rangle 3$. $checkpointedHistorySummary \in HistorySummaryType$

$\langle 2\rangle 1$. $BaseHashValue \in HashType$

BY *BaseHashValueTypeSafe*

$\langle 2\rangle 2$. QED

BY $\langle 1\rangle 2$, $\langle 2\rangle 1$ DEF *checkpointedHistorySummary*, *HistorySummaryType*

$\langle 1\rangle 4$. $\land$ $checkpointedHistorySummary.anchor \in HashType$

$\land$ $checkpointedHistorySummary.extension \in HashType$

BY $\langle 1\rangle 3$ DEF *HistorySummaryType*

$\langle 1\rangle 5$. QED

BY $\langle 1\rangle 2$, $\langle 1\rangle 3$, $\langle 1\rangle 4$ DEF *checkpointedAnchor*, *checkpointedHistorySummary*

Type safety of the *Checkpoint* function.

THEOREM $CheckpointTypeSafe \triangleq$

$\forall historySummary \in HistorySummaryType : Checkpoint(historySummary) \in HistorySummaryType$

$\langle 1\rangle 1$. TAKE $historySummary \in HistorySummaryType$

$\langle 1\rangle$ $checkpointedAnchor \triangleq Hash(historySummary.anchor, historySummary.extension)$

$\langle 1\rangle$ $checkpointedHistorySummary \triangleq [$

$anchor \mapsto checkpointedAnchor,$

$extension \mapsto BaseHashValue]$

$\langle 1\rangle 2$. $\land$ $checkpointedAnchor \in HashType$

$\land$ $checkpointedHistorySummary \in HistorySummaryType$

$\land$ $checkpointedHistorySummary.anchor \in HashType$

$\land$ $checkpointedHistorySummary.extension \in HashType$

BY $\langle 1\rangle 1$, *CheckpointDefsTypeSafeLemma*

$\langle 1\rangle$ HIDE DEF *checkpointedAnchor*, *checkpointedHistorySummary*

$\langle 1\rangle 3$. $\lor$ $Checkpoint(historySummary) = historySummary$

$\lor$ $Checkpoint(historySummary) = checkpointedHistorySummary$

BY $\langle 1\rangle 1$ DEF *Checkpoint*, *checkpointedHistorySummary*, *checkpointedAnchor*

$\langle 1\rangle 4$. $historySummary \in HistorySummaryType$

BY $\langle 1\rangle 1$

$\langle 1\rangle 5$. $checkpointedHistorySummary \in HistorySummaryType$

BY $\langle 1\rangle 2$

$\langle 1\rangle 6$. QED

BY $\langle 1\rangle 3$, $\langle 1\rangle 4$, $\langle 1\rangle 5$

The *SuccessorDefsTypeSafeLemma* proves that the definitions within the LET of the *Successor* function all have the appropriate type. This is a trivial proof that merely walks through the definitions.

THEOREM $SuccessorDefsTypeSafeLemma \triangleq$

$\forall historySummary \in HistorySummaryType, input \in InputType, hashBarrier \in HashType :$

LET

$securedInput \triangleq Hash(hashBarrier, input)$

$newAnchor \triangleq historySummary.anchor$

$newExtension \triangleq Hash(historySummary.extension, securedInput)$

$newHistorySummary \triangleq [$

$anchor \mapsto newAnchor,$

$extension \mapsto newExtension]$

IN

$\land$ $securedInput \in HashType$

$\land$ $newAnchor \in HashType$

$\land$ $newExtension \in HashType$

$\land\ \mathit{newHistorySummary} \in \mathit{HistorySummaryType}$
$\land\ \mathit{newHistorySummary.anchor} \in \mathit{HashType}$
$\land\ \mathit{newHistorySummary.extension} \in \mathit{HashType}$
$\langle 1\rangle 1.$ TAKE $\mathit{historySummary} \in \mathit{HistorySummaryType},\ \mathit{input} \in \mathit{InputType},\ \mathit{hashBarrier} \in \mathit{HashType}$
$\langle 1\rangle\ \mathit{securedInput} \triangleq \mathit{Hash}(\mathit{hashBarrier},\ \mathit{input})$
$\langle 1\rangle\ \mathit{newAnchor} \triangleq \mathit{historySummary.anchor}$
$\langle 1\rangle\ \mathit{newExtension} \triangleq \mathit{Hash}(\mathit{historySummary.extension},\ \mathit{securedInput})$
$\langle 1\rangle\ \mathit{newHistorySummary} \triangleq [$
$\mathit{anchor} \mapsto \mathit{newAnchor},$
$\mathit{extension} \mapsto \mathit{newExtension}]$
$\langle 1\rangle$ HIDE DEF $\mathit{securedInput},\ \mathit{newAnchor},\ \mathit{newExtension},\ \mathit{newHistorySummary}$
$\langle 1\rangle 2.\ \mathit{securedInput} \in \mathit{HashType}$
$\langle 2\rangle 1.\ \mathit{hashBarrier} \in \mathit{HashDomain}$
$\langle 3\rangle 1.\ \mathit{hashBarrier} \in \mathit{HashType}$
BY $\langle 1\rangle 1$
$\langle 3\rangle 2.$ QED
BY $\langle 3\rangle 1$ DEF $\mathit{HashDomain}$
$\langle 2\rangle 2.\ \mathit{input} \in \mathit{HashDomain}$
$\langle 3\rangle 1.\ \mathit{input} \in \mathit{InputType}$
BY $\langle 1\rangle 1$
$\langle 3\rangle 2.$ QED
BY $\langle 3\rangle 1$ DEF $\mathit{HashDomain}$
$\langle 2\rangle 3.$ QED
BY $\langle 2\rangle 1,\ \langle 2\rangle 2,\ \mathit{HashTypeSafe}$DEF $\mathit{securedInput}$
$\langle 1\rangle 3.\ \mathit{newAnchor} \in \mathit{HashType}$
$\langle 2\rangle 1.\ \mathit{historySummary} \in \mathit{HistorySummaryType}$
BY $\langle 1\rangle 1$
$\langle 2\rangle 2.$ QED
BY $\langle 2\rangle 1$ DEF $\mathit{newAnchor},\ \mathit{HistorySummaryType}$
$\langle 1\rangle 4.\ \mathit{newExtension} \in \mathit{HashType}$
$\langle 2\rangle 1.\ \mathit{historySummary.extension} \in \mathit{HashDomain}$
$\langle 3\rangle 1.\ \mathit{historySummary.extension} \in \mathit{HashType}$
$\langle 4\rangle 1.\ \mathit{historySummary} \in \mathit{HistorySummaryType}$
BY $\langle 1\rangle 1$
$\langle 4\rangle 2.$ QED
BY $\langle 4\rangle 1$ DEF $\mathit{HistorySummaryType}$
$\langle 3\rangle 2.$ QED
BY $\langle 3\rangle 1$ DEF $\mathit{HashDomain}$
$\langle 2\rangle 2.\ \mathit{securedInput} \in \mathit{HashDomain}$
BY $\langle 1\rangle 2$ DEF $\mathit{HashDomain}$
$\langle 2\rangle 3.$ QED
BY $\langle 2\rangle 1,\ \langle 2\rangle 2,\ \mathit{HashTypeSafe}$DEF $\mathit{newExtension}$
$\langle 1\rangle 5.\ \mathit{newHistorySummary} \in \mathit{HistorySummaryType}$
BY $\langle 1\rangle 3,\ \langle 1\rangle 4$ DEF $\mathit{newHistorySummary},\ \mathit{HistorySummaryType}$
$\langle 1\rangle 6.\ \land\ \mathit{newHistorySummary.anchor} \in \mathit{HashType}$
$\land\ \mathit{newHistorySummary.extension} \in \mathit{HashType}$
BY $\langle 1\rangle 5$ DEF $\mathit{HistorySummaryType}$
$\langle 1\rangle 7.$ QED
BY $\langle 1\rangle 2,\ \langle 1\rangle 3,\ \langle 1\rangle 4,\ \langle 1\rangle 5,\ \langle 1\rangle 6$
DEF $\mathit{securedInput},\ \mathit{newAnchor},\ \mathit{newExtension},\ \mathit{newHistorySummary}$

Type safety of the *Successor* function.

THEOREM $SuccessorTypeSafe \triangleq$

$\quad \forall\, historySummary \in HistorySummaryType,\ input \in InputType,\ hashBarrier \in HashType :$

$\qquad Successor(historySummary,\ input,\ hashBarrier) \in HistorySummaryType$

$\langle 1 \rangle 1.$ TAKE $historySummary \in HistorySummaryType,\ input \in InputType,\ hashBarrier \in HashType$

$\langle 1 \rangle\ securedInput \triangleq Hash(hashBarrier,\ input)$

$\langle 1 \rangle\ newAnchor \triangleq historySummary.anchor$

$\langle 1 \rangle\ newExtension \triangleq Hash(historySummary.extension,\ securedInput)$

$\langle 1 \rangle\ newHistorySummary \triangleq [$

$\qquad anchor \mapsto newAnchor,$

$\qquad extension \mapsto newExtension]$

$\langle 1 \rangle 2.\ \land\ securedInput \in HashType$

$\qquad \land\ newAnchor \in HashType$

$\qquad \land\ newExtension \in HashType$

$\qquad \land\ newHistorySummary \in HistorySummaryType$

$\qquad \land\ newHistorySummary.anchor \in HashType$

$\qquad \land\ newHistorySummary.extension \in HashType$

$\quad$ BY $\langle 1 \rangle 1,\ SuccessorDefsTypeSafeLemma$

$\langle 1 \rangle$ HIDE DEF $securedInput,\ newAnchor,\ newExtension,\ newHistorySummary$

$\langle 1 \rangle 3.\ Successor(historySummary,\ input,\ hashBarrier) = newHistorySummary$

$\quad$ BY $\langle 1 \rangle 1$ DEF $Successor,\ newHistorySummary,\ newExtension,\ newAnchor,\ securedInput$

$\langle 1 \rangle 4.\ newHistorySummary \in HistorySummaryType$

$\quad$ BY $\langle 1 \rangle 2$

$\langle 1 \rangle 5.$ QED

$\quad$ BY $\langle 1 \rangle 3,\ \langle 1 \rangle 4$

The *LL2SubtypeImplicationLemma* proves that when the *LL2TypeInvariant* holds, the subtypes of *LL2Disk*, *LL2RAM*, and *LL2NVRAM* also hold. This is asserted and proven for both the unprimed and primed states.

The proof itself is completely trivial. It follows directly from the type definitions *LL2UntrustedStorageType* and *LL2TrustedStorageType*.

$LL2SubtypeImplication \triangleq$

$\quad LL2TypeInvariant \Rightarrow$

$\qquad \land\ LL2Disk.publicState \in PublicStateType$

$\qquad \land\ LL2Disk.privateStateEnc \in PrivateStateEncType$

$\qquad \land\ LL2Disk.historySummary \in HistorySummaryType$

$\qquad \land\ LL2Disk.historySummary.anchor \in HashType$

$\qquad \land\ LL2Disk.historySummary.extension \in HashType$

$\qquad \land\ LL2Disk.authenticator \in MACType$

$\qquad \land\ LL2RAM.publicState \in PublicStateType$

$\qquad \land\ LL2RAM.privateStateEnc \in PrivateStateEncType$

$\qquad \land\ LL2RAM.historySummary \in HistorySummaryType$

$\qquad \land\ LL2RAM.historySummary.anchor \in HashType$

$\qquad \land\ LL2RAM.historySummary.extension \in HashType$

$\qquad \land\ LL2RAM.authenticator \in MACType$

$\qquad \land\ LL2NVRAM.historySummaryAnchor \in HashType$

$\qquad \land\ LL2NVRAM.symmetricKey \in SymmetricKeyType$

$\qquad \land\ LL2NVRAM.hashBarrier \in HashType$

$\qquad \land\ LL2NVRAM.extensionInProgress \in$ BOOLEAN

THEOREM $LL2SubtypeImplicationLemma \triangleq$

$\quad \land\ LL2SubtypeImplication$

$\quad \land\ LL2SubtypeImplication'$

$\langle 1 \rangle 1.\ LL2SubtypeImplication$

$\langle 2 \rangle 1.$ SUFFICES
ASSUME $LL2TypeInvariant$
PROVE
$\land$ $LL2Disk.publicState \in PublicStateType$
$\land$ $LL2Disk.privateStateEnc \in PrivateStateEncType$
$\land$ $LL2Disk.historySummary \in HistorySummaryType$
$\land$ $LL2Disk.historySummary.anchor \in HashType$
$\land$ $LL2Disk.historySummary.extension \in HashType$
$\land$ $LL2Disk.authenticator \in MACType$
$\land$ $LL2RAM.publicState \in PublicStateType$
$\land$ $LL2RAM.privateStateEnc \in PrivateStateEncType$
$\land$ $LL2RAM.historySummary \in HistorySummaryType$
$\land$ $LL2RAM.historySummary.anchor \in HashType$
$\land$ $LL2RAM.historySummary.extension \in HashType$
$\land$ $LL2RAM.authenticator \in MACType$
$\land$ $LL2NVRAM.historySummaryAnchor \in HashType$
$\land$ $LL2NVRAM.symmetricKey \in SymmetricKeyType$
$\land$ $LL2NVRAM.hashBarrier \in HashType$
$\land$ $LL2NVRAM.extensionInProgress \in$ BOOLEAN
BY DEF $LL2SubtypeImplication$, $LL2TypeInvariant$
$\langle 2 \rangle 2.$ $LL2Disk \in LL2UntrustedStorageType$
BY $\langle 2 \rangle 1$ DEF $LL2TypeInvariant$
$\langle 2 \rangle 3.$ $LL2RAM \in LL2UntrustedStorageType$
BY $\langle 2 \rangle 1$ DEF $LL2TypeInvariant$
$\langle 2 \rangle 4.$ $LL2NVRAM \in LL2TrustedStorageType$
BY $\langle 2 \rangle 1$ DEF $LL2TypeInvariant$
$\langle 2 \rangle 5.$ $LL2Disk.publicState \in PublicStateType$
BY $\langle 2 \rangle 2$ DEF $LL2UntrustedStorageType$
$\langle 2 \rangle 6.$ $LL2Disk.privateStateEnc \in PrivateStateEncType$
BY $\langle 2 \rangle 2$ DEF $LL2UntrustedStorageType$
$\langle 2 \rangle 7.$ $LL2Disk.historySummary \in HistorySummaryType$
BY $\langle 2 \rangle 2$ DEF $LL2UntrustedStorageType$
$\langle 2 \rangle 8.$ $LL2Disk.historySummary.anchor \in HashType$
BY $\langle 2 \rangle 7$ DEF $HistorySummaryType$
$\langle 2 \rangle 9.$ $LL2Disk.historySummary.extension \in HashType$
BY $\langle 2 \rangle 7$ DEF $HistorySummaryType$
$\langle 2 \rangle 10.$ $LL2Disk.authenticator \in MACType$
BY $\langle 2 \rangle 2$ DEF $LL2UntrustedStorageType$
$\langle 2 \rangle 11.$ $LL2RAM.publicState \in PublicStateType$
BY $\langle 2 \rangle 3$ DEF $LL2UntrustedStorageType$
$\langle 2 \rangle 12.$ $LL2RAM.privateStateEnc \in PrivateStateEncType$
BY $\langle 2 \rangle 3$ DEF $LL2UntrustedStorageType$
$\langle 2 \rangle 13.$ $LL2RAM.historySummary \in HistorySummaryType$
BY $\langle 2 \rangle 3$ DEF $LL2UntrustedStorageType$
$\langle 2 \rangle 14.$ $LL2RAM.historySummary.anchor \in HashType$
BY $\langle 2 \rangle 13$ DEF $HistorySummaryType$
$\langle 2 \rangle 15.$ $LL2RAM.historySummary.extension \in HashType$
BY $\langle 2 \rangle 13$ DEF $HistorySummaryType$
$\langle 2 \rangle 16.$ $LL2RAM.authenticator \in MACType$
BY $\langle 2 \rangle 3$ DEF $LL2UntrustedStorageType$
$\langle 2 \rangle 17.$ $LL2NVRAM.historySummaryAnchor \in HashType$
BY $\langle 2 \rangle 4$ DEF $LL2TrustedStorageType$

$\langle 2 \rangle 18.$ $LL2NVRAM.symmetricKey \in SymmetricKeyType$
BY $\langle 2 \rangle 4$ DEF $LL2TrustedStorageType$
$\langle 2 \rangle 19.$ $LL2NVRAM.hashBarrier \in HashType$
BY $\langle 2 \rangle 4$ DEF $LL2TrustedStorageType$
$\langle 2 \rangle 20.$ $LL2NVRAM.extensionInProgress \in \text{BOOLEAN}$
BY $\langle 2 \rangle 4$ DEF $LL2TrustedStorageType$
$\langle 2 \rangle 21.$ QED
BY $\langle 2 \rangle 5$, $\langle 2 \rangle 6$, $\langle 2 \rangle 7$, $\langle 2 \rangle 8$, $\langle 2 \rangle 9$, $\langle 2 \rangle 10$, $\langle 2 \rangle 11$, $\langle 2 \rangle 12$, $\langle 2 \rangle 13$, $\langle 2 \rangle 14$, $\langle 2 \rangle 15$, $\langle 2 \rangle 16$, $\langle 2 \rangle 17$, $\langle 2 \rangle 18$, $\langle 2 \rangle 19$, $\langle 2 \rangle 20$
$\langle 1 \rangle 2.$ $LL2SubtypeImplication'$
$\langle 2 \rangle 1.$ SUFFICES
ASSUME $LL2TypeInvariant'$
PROVE
$\land$ $LL2Disk.publicState' \in PublicStateType$
$\land$ $LL2Disk.privateStateEnc' \in PrivateStateEncType$
$\land$ $LL2Disk.historySummary' \in HistorySummaryType$
$\land$ $LL2Disk.historySummary.anchor' \in HashType$
$\land$ $LL2Disk.historySummary.extension' \in HashType$
$\land$ $LL2Disk.authenticator' \in MACType$
$\land$ $LL2RAM.publicState' \in PublicStateType$
$\land$ $LL2RAM.privateStateEnc' \in PrivateStateEncType$
$\land$ $LL2RAM.historySummary' \in HistorySummaryType$
$\land$ $LL2RAM.historySummary.anchor' \in HashType$
$\land$ $LL2RAM.historySummary.extension' \in HashType$
$\land$ $LL2RAM.authenticator' \in MACType$
$\land$ $LL2NVRAM.historySummaryAnchor' \in HashType$
$\land$ $LL2NVRAM.symmetricKey' \in SymmetricKeyType$
$\land$ $LL2NVRAM.hashBarrier' \in HashType$
$\land$ $LL2NVRAM.extensionInProgress' \in \text{BOOLEAN}$
BY DEF $LL2SubtypeImplication$, $LL2TypeInvariant$
$\langle 2 \rangle 2.$ $LL2Disk' \in LL2UntrustedStorageType$
BY $\langle 2 \rangle 1$ DEF $LL2TypeInvariant$
$\langle 2 \rangle 3.$ $LL2RAM' \in LL2UntrustedStorageType$
BY $\langle 2 \rangle 1$ DEF $LL2TypeInvariant$
$\langle 2 \rangle 4.$ $LL2NVRAM' \in LL2TrustedStorageType$
BY $\langle 2 \rangle 1$ DEF $LL2TypeInvariant$
$\langle 2 \rangle 5.$ $LL2Disk.publicState' \in PublicStateType$
BY $\langle 2 \rangle 2$ DEF $LL2UntrustedStorageType$
$\langle 2 \rangle 6.$ $LL2Disk.privateStateEnc' \in PrivateStateEncType$
BY $\langle 2 \rangle 2$ DEF $LL2UntrustedStorageType$
$\langle 2 \rangle 7.$ $LL2Disk.historySummary' \in HistorySummaryType$
BY $\langle 2 \rangle 2$ DEF $LL2UntrustedStorageType$
$\langle 2 \rangle 8.$ $LL2Disk.historySummary.anchor' \in HashType$
BY $\langle 2 \rangle 7$ DEF $HistorySummaryType$
$\langle 2 \rangle 9.$ $LL2Disk.historySummary.extension' \in HashType$
BY $\langle 2 \rangle 7$ DEF $HistorySummaryType$
$\langle 2 \rangle 10.$ $LL2Disk.authenticator' \in MACType$
BY $\langle 2 \rangle 2$ DEF $LL2UntrustedStorageType$
$\langle 2 \rangle 11.$ $LL2RAM.publicState' \in PublicStateType$
BY $\langle 2 \rangle 3$ DEF $LL2UntrustedStorageType$
$\langle 2 \rangle 12.$ $LL2RAM.privateStateEnc' \in PrivateStateEncType$
BY $\langle 2 \rangle 3$ DEF $LL2UntrustedStorageType$

$\langle 2\rangle 13.$ $LL2RAM.historySummary' \in HistorySummaryType$
BY $\langle 2\rangle 3$ DEF $LL2UntrustedStorageType$
$\langle 2\rangle 14.$ $LL2RAM.historySummary.anchor' \in HashType$
BY $\langle 2\rangle 13$ DEF $HistorySummaryType$
$\langle 2\rangle 15.$ $LL2RAM.historySummary.extension' \in HashType$
BY $\langle 2\rangle 13$ DEF $HistorySummaryType$
$\langle 2\rangle 16.$ $LL2RAM.authenticator' \in MACType$
BY $\langle 2\rangle 3$ DEF $LL2UntrustedStorageType$
$\langle 2\rangle 17.$ $LL2NVRAM.historySummaryAnchor' \in HashType$
BY $\langle 2\rangle 4$ DEF $LL2TrustedStorageType$
$\langle 2\rangle 18.$ $LL2NVRAM.symmetricKey' \in SymmetricKeyType$
BY $\langle 2\rangle 4$ DEF $LL2TrustedStorageType$
$\langle 2\rangle 19.$ $LL2NVRAM.hashBarrier' \in HashType$
BY $\langle 2\rangle 4$ DEF $LL2TrustedStorageType$
$\langle 2\rangle 20.$ $LL2NVRAM.extensionInProgress' \in$ BOOLEAN
BY $\langle 2\rangle 4$ DEF $LL2TrustedStorageType$
$\langle 2\rangle 21.$ QED
BY $\langle 2\rangle 5$, $\langle 2\rangle 6$, $\langle 2\rangle 7$, $\langle 2\rangle 8$, $\langle 2\rangle 9$, $\langle 2\rangle 10$, $\langle 2\rangle 11$, $\langle 2\rangle 12$,
$\langle 2\rangle 13$, $\langle 2\rangle 14$, $\langle 2\rangle 15$, $\langle 2\rangle 16$, $\langle 2\rangle 17$, $\langle 2\rangle 18$, $\langle 2\rangle 19$, $\langle 2\rangle 20$
$\langle 1\rangle 3.$ QED
BY $\langle 1\rangle 1$, $\langle 1\rangle 2$

The *LL2InitDefsTypeSafeLemma* proves that the definitions within the LET of the *LL2Init* action all have the appropriate type. This is a trivial proof that merely walks through the definitions.

THEOREM $LL2InitDefsTypeSafeLemma \triangleq$
$\forall\, symmetricKey \in SymmetricKeyType,\ hashBarrier \in HashType :$
LET
$initialPrivateStateEnc \triangleq SymmetricEncrypt(symmetricKey, InitialPrivateState)$
$initialStateHash \triangleq Hash(InitialPublicState, initialPrivateStateEnc)$
$initialHistorySummary \triangleq [$
$anchor \mapsto BaseHashValue,$
$extension \mapsto BaseHashValue]$
$initialHistorySummaryHash \triangleq Hash(BaseHashValue, BaseHashValue)$
$initialHistoryStateBinding \triangleq Hash(initialHistorySummaryHash, initialStateHash)$
$initialAuthenticator \triangleq GenerateMAC(symmetricKey, initialHistoryStateBinding)$
$initialUntrustedStorage \triangleq [$
$publicState \mapsto InitialPublicState,$
$privateStateEnc \mapsto initialPrivateStateEnc,$
$historySummary \mapsto initialHistorySummary,$
$authenticator \mapsto initialAuthenticator]$
$initialTrustedStorage \triangleq [$
$historySummaryAnchor \mapsto BaseHashValue,$
$symmetricKey \mapsto symmetricKey,$
$hashBarrier \mapsto hashBarrier,$
$extensionInProgress \mapsto$ FALSE$]$
IN
$\land\ initialPrivateStateEnc \in PrivateStateEncType$
$\land\ initialStateHash \in HashType$
$\land\ initialHistorySummary \in HistorySummaryType$
$\land\ initialHistorySummaryHash \in HashType$
$\land\ initialHistoryStateBinding \in HashType$
$\land\ initialAuthenticator \in MACType$

$\land\ initialUntrustedStorage \in LL2UntrustedStorageType$
$\land\ initialTrustedStorage \in LL2TrustedStorageType$
$\langle 1\rangle 1.$ TAKE $symmetricKey \in SymmetricKeyType$, $hashBarrier \in HashType$
$\langle 1\rangle$ $initialPrivateStateEnc \triangleq SymmetricEncrypt(symmetricKey, InitialPrivateState)$
$\langle 1\rangle$ $initialStateHash \triangleq Hash(InitialPublicState, initialPrivateStateEnc)$
$\langle 1\rangle$ $initialHistorySummary \triangleq [$
$anchor \mapsto BaseHashValue,$
$extension \mapsto BaseHashValue]$
$\langle 1\rangle$ $initialHistorySummaryHash \triangleq Hash(BaseHashValue, BaseHashValue)$
$\langle 1\rangle$ $initialHistoryStateBinding \triangleq Hash(initialHistorySummaryHash, initialStateHash)$
$\langle 1\rangle$ $initialAuthenticator \triangleq GenerateMAC(symmetricKey, initialHistoryStateBinding)$
$\langle 1\rangle$ $initialUntrustedStorage \triangleq [$
$publicState \mapsto InitialPublicState,$
$privateStateEnc \mapsto initialPrivateStateEnc,$
$historySummary \mapsto initialHistorySummary,$
$authenticator \mapsto initialAuthenticator]$
$\langle 1\rangle$ $initialTrustedStorage \triangleq [$
$historySummaryAnchor \mapsto BaseHashValue,$
$symmetricKey \mapsto symmetricKey,$
$hashBarrier \mapsto hashBarrier,$
$extensionInProgress \mapsto$ FALSE$]$
$\langle 1\rangle$ HIDE DEF $initialPrivateStateEnc$, $initialStateHash$, $initialHistorySummary$,
$initialHistorySummaryHash$, $initialHistoryStateBinding$, $initialAuthenticator$,
$initialUntrustedStorage$, $initialTrustedStorage$
$\langle 1\rangle 2.$ $initialPrivateStateEnc \in PrivateStateEncType$
$\langle 2\rangle 1.$ $symmetricKey \in SymmetricKeyType$
BY $\langle 1\rangle 1$
$\langle 2\rangle 2.$ $InitialPrivateState \in PrivateStateType$
BY $ConstantsTypeSafe$
$\langle 2\rangle 3.$ QED
BY $\langle 2\rangle 1$, $\langle 2\rangle 2$, $SymmetricEncryptionTypeSafe$DEF $initialPrivateStateEnc$
$\langle 1\rangle 3.$ $initialStateHash \in HashType$
$\langle 2\rangle 1.$ $InitialPublicState \in HashDomain$
$\langle 3\rangle 1.$ $InitialPublicState \in PublicStateType$
BY $ConstantsTypeSafe$
$\langle 3\rangle 2.$ QED
BY $\langle 3\rangle 1$ DEF $HashDomain$
$\langle 2\rangle 2.$ $initialPrivateStateEnc \in HashDomain$
BY $\langle 1\rangle 2$ DEF $HashDomain$
$\langle 2\rangle 3.$ QED
BY $\langle 2\rangle 1$, $\langle 2\rangle 2$, $HashTypeSafe$DEF $initialStateHash$
$\langle 1\rangle 4.$ $initialHistorySummary \in HistorySummaryType$
$\langle 2\rangle 1.$ $BaseHashValue \in HashType$
BY $BaseHashValueTypeSafe$
$\langle 2\rangle 2.$ QED
BY $\langle 2\rangle 1$ DEF $initialHistorySummary$, $HistorySummaryType$
$\langle 1\rangle 5.$ $initialHistorySummaryHash \in HashType$
$\langle 2\rangle 1.$ $BaseHashValue \in HashDomain$
$\langle 3\rangle 1.$ $BaseHashValue \in HashType$
BY $BaseHashValueTypeSafe$
$\langle 3\rangle 2.$ QED
BY $\langle 3\rangle 1$ DEF $HashDomain$

$\langle 2\rangle 2.$ QED
BY $\langle 2\rangle 1$, *HashTypeSafe*DEF *initialHistorySummaryHash*
$\langle 1\rangle 6.$ *initialHistoryStateBinding* $\in$ *HashType*
$\langle 2\rangle 1.$ *initialHistorySummaryHash* $\in$ *HashDomain*
BY $\langle 1\rangle 5$ DEF *HashDomain*
$\langle 2\rangle 2.$ *initialStateHash* $\in$ *HashDomain*
BY $\langle 1\rangle 3$ DEF *HashDomain*
$\langle 2\rangle 3.$ QED
BY $\langle 2\rangle 1$, $\langle 2\rangle 2$, *HashTypeSafe*DEF *initialHistoryStateBinding*
$\langle 1\rangle 7.$ *initialAuthenticator* $\in$ *MACType*
$\langle 2\rangle 1.$ *symmetricKey* $\in$ *SymmetricKeyType*
BY $\langle 1\rangle 1$
$\langle 2\rangle 2.$ *initialHistoryStateBinding* $\in$ *HashType*
BY $\langle 1\rangle 6$
$\langle 2\rangle 3.$ QED
BY $\langle 2\rangle 1$, $\langle 2\rangle 2$, *GenerateMACTypeSafe*DEF *initialAuthenticator*
$\langle 1\rangle 8.$ *initialUntrustedStorage* $\in$ *LL2UntrustedStorageType*
$\langle 2\rangle 1.$ *InitialPublicState* $\in$ *PublicStateType*
BY *ConstantsTypeSafe*
$\langle 2\rangle 2.$ *initialPrivateStateEnc* $\in$ *PrivateStateEncType*
BY $\langle 1\rangle 2$
$\langle 2\rangle 3.$ *initialHistorySummary* $\in$ *HistorySummaryType*
BY $\langle 1\rangle 4$
$\langle 2\rangle 4.$ *initialAuthenticator* $\in$ *MACType*
BY $\langle 1\rangle 7$
$\langle 2\rangle 5.$ QED
BY $\langle 2\rangle 1$, $\langle 2\rangle 2$, $\langle 2\rangle 3$, $\langle 2\rangle 4$ DEF *initialUntrustedStorage*, *LL2UntrustedStorageType*
$\langle 1\rangle 9.$ *initialTrustedStorage* $\in$ *LL2TrustedStorageType*
$\langle 2\rangle 1.$ *BaseHashValue* $\in$ *HashType*
BY *BaseHashValueTypeSafe*
$\langle 2\rangle 2.$ *symmetricKey* $\in$ *SymmetricKeyType*
BY $\langle 1\rangle 1$
$\langle 2\rangle 3.$ *hashBarrier* $\in$ *HashType*
BY $\langle 1\rangle 1$
$\langle 2\rangle 4.$ FALSE $\in$ BOOLEAN
OBVIOUS
$\langle 2\rangle 5.$ QED
BY $\langle 2\rangle 1$, $\langle 2\rangle 2$, $\langle 2\rangle 3$, $\langle 2\rangle 4$ DEF *initialTrustedStorage*, *LL2TrustedStorageType*
$\langle 1\rangle 10.$ QED
BY $\langle 1\rangle 2$, $\langle 1\rangle 3$, $\langle 1\rangle 4$, $\langle 1\rangle 5$, $\langle 1\rangle 6$, $\langle 1\rangle 7$, $\langle 1\rangle 8$, $\langle 1\rangle 9$
DEF *initialPrivateStateEnc*, *initialStateHash*, *initialHistorySummary*,
initialHistorySummaryHash, *initialHistoryStateBinding*, *initialAuthenticator*,
initialUntrustedStorage, *initialTrustedStorage*

The *LL2PerformOperationDefsTypeSafeLemma* proves that the definitions within the LET of the *LL2PerformOperation* action all have the appropriate type. This is a trivial proof that merely walks through the definitions.

THEOREM *LL2PerformOperationDefsTypeSafeLemma* $\triangleq$
$\forall$ *input* $\in$ *LL2AvailableInputs* :
LL2TypeInvariant $\Rightarrow$
LET
historySummaryHash $\triangleq$
Hash(*LL2RAM.historySummary.anchor*, *LL2RAM.historySummary.extension*)

$stateHash \triangleq Hash(LL2RAM.publicState, LL2RAM.privateStateEnc)$
$historyStateBinding \triangleq Hash(historySummaryHash, stateHash)$
$privateState \triangleq SymmetricDecrypt(LL2NVRAM.symmetricKey, LL2RAM.privateStateEnc)$
$sResult \triangleq Service(LL2RAM.publicState, privateState, input)$
$newPrivateStateEnc \triangleq$
$SymmetricEncrypt(LL2NVRAM.symmetricKey, sResult.newPrivateState)$
$currentHistorySummary \triangleq [$
$anchor \mapsto LL2NVRAM.historySummaryAnchor,$
$extension \mapsto LL2SPCR]$
$newHistorySummary \triangleq Successor(currentHistorySummary, input, LL2NVRAM.hashBarrier)$
$newHistorySummaryHash \triangleq$
$Hash(newHistorySummary.anchor, newHistorySummary.extension)$
$newStateHash \triangleq Hash(sResult.newPublicState, newPrivateStateEnc)$
$newHistoryStateBinding \triangleq Hash(newHistorySummaryHash, newStateHash)$
$newAuthenticator \triangleq GenerateMAC(LL2NVRAM.symmetricKey, newHistoryStateBinding)$
IN
$\land\ historySummaryHash \in HashType$
$\land\ stateHash \in HashType$
$\land\ historyStateBinding \in HashType$
$\land\ privateState \in PrivateStateType$
$\land\ sResult \in ServiceResultType$
$\land\ sResult.newPublicState \in PublicStateType$
$\land\ sResult.newPrivateState \in PrivateStateType$
$\land\ sResult.output \in OutputType$
$\land\ newPrivateStateEnc \in PrivateStateEncType$
$\land\ currentHistorySummary \in HistorySummaryType$
$\land\ currentHistorySummary.anchor \in HashType$
$\land\ currentHistorySummary.extension \in HashType$
$\land\ newHistorySummary \in HistorySummaryType$
$\land\ newHistorySummary.anchor \in HashType$
$\land\ newHistorySummary.extension \in HashType$
$\land\ newHistorySummaryHash \in HashType$
$\land\ newStateHash \in HashType$
$\land\ newHistoryStateBinding \in HashType$
$\land\ newAuthenticator \in MACType$
$\langle 1 \rangle 1.$ TAKE $input \in LL2AvailableInputs$
$\langle 1 \rangle\ historySummaryHash \triangleq Hash(LL2RAM.historySummary.anchor, LL2RAM.historySummary.extension)$
$\langle 1 \rangle\ stateHash \triangleq Hash(LL2RAM.publicState, LL2RAM.privateStateEnc)$
$\langle 1 \rangle\ historyStateBinding \triangleq Hash(historySummaryHash, stateHash)$
$\langle 1 \rangle\ privateState \triangleq SymmetricDecrypt(LL2NVRAM.symmetricKey, LL2RAM.privateStateEnc)$
$\langle 1 \rangle\ sResult \triangleq Service(LL2RAM.publicState, privateState, input)$
$\langle 1 \rangle\ newPrivateStateEnc \triangleq$
$SymmetricEncrypt(LL2NVRAM.symmetricKey, sResult.newPrivateState)$
$\langle 1 \rangle\ currentHistorySummary \triangleq [$
$anchor \mapsto LL2NVRAM.historySummaryAnchor,$
$extension \mapsto LL2SPCR]$
$\langle 1 \rangle\ newHistorySummary \triangleq Successor(currentHistorySummary, input, LL2NVRAM.hashBarrier)$
$\langle 1 \rangle\ newHistorySummaryHash \triangleq Hash(newHistorySummary.anchor, newHistorySummary.extension)$
$\langle 1 \rangle\ newStateHash \triangleq Hash(sResult.newPublicState, newPrivateStateEnc)$
$\langle 1 \rangle\ newHistoryStateBinding \triangleq Hash(newHistorySummaryHash, newStateHash)$
$\langle 1 \rangle\ newAuthenticator \triangleq GenerateMAC(LL2NVRAM.symmetricKey, newHistoryStateBinding)$
$\langle 1 \rangle$ HIDE DEF $historySummaryHash, stateHash, historyStateBinding, privateState,$

sResult, *newPrivateStateEnc*, *currentHistorySummary*, *newHistorySummary*,
newHistorySummaryHash, *newStateHash*, *newHistoryStateBinding*, *newAuthenticator*

⟨1⟩2. HAVE *LL2TypeInvariant*
⟨1⟩3. $historySummaryHash \in HashType$
⟨2⟩1. $LL2RAM.historySummary.anchor \in HashDomain$
⟨3⟩1. $LL2RAM.historySummary.anchor \in HashType$
BY ⟨1⟩2, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*
⟨3⟩2. QED
BY ⟨3⟩1 DEF *HashDomain*
⟨2⟩2. $LL2RAM.historySummary.extension \in HashDomain$
⟨3⟩1. $LL2RAM.historySummary.extension \in HashType$
BY ⟨1⟩2, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*
⟨3⟩2. QED
BY ⟨3⟩1 DEF *HashDomain*
⟨2⟩3. QED
BY ⟨2⟩1, ⟨2⟩2, *HashTypeSafe*DEF *historySummaryHash*
⟨1⟩4. $stateHash \in HashType$
⟨2⟩1. $\land$ $LL2RAM.publicState \in PublicStateType$
$\land$ $LL2RAM.privateStateEnc \in PrivateStateEncType$
BY ⟨1⟩2, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*
⟨2⟩2. $\land$ $LL2RAM.publicState \in HashDomain$
$\land$ $LL2RAM.privateStateEnc \in HashDomain$
BY ⟨2⟩1 DEF *HashDomain*
⟨2⟩3. QED
BY ⟨2⟩2, *HashTypeSafe*DEF *stateHash*
⟨1⟩5. $historyStateBinding \in HashType$
⟨2⟩1. $historySummaryHash \in HashDomain$
BY ⟨1⟩3 DEF *HashDomain*
⟨2⟩2. $stateHash \in HashDomain$
BY ⟨1⟩4 DEF *HashDomain*
⟨2⟩3. QED
BY ⟨2⟩1, ⟨2⟩2, *HashTypeSafe*DEF *historyStateBinding*
⟨1⟩6. $privateState \in PrivateStateType$
⟨2⟩1. $\land$ $LL2NVRAM.symmetricKey \in SymmetricKeyType$
$\land$ $LL2RAM.privateStateEnc \in PrivateStateEncType$
BY ⟨1⟩2, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*
⟨2⟩2. QED
BY ⟨2⟩1, *SymmetricDecryptionTypeSafe*DEF *privateState*
⟨1⟩7. $sResult \in ServiceResultType$
⟨2⟩1. $LL2RAM.publicState \in PublicStateType$
BY ⟨1⟩2, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*
⟨2⟩2. $privateState \in PrivateStateType$
BY ⟨1⟩6
⟨2⟩3. $input \in InputType$
⟨3⟩1. $LL2AvailableInputs \subseteq InputType$
BY ⟨1⟩2 DEF *LL2TypeInvariant*
⟨3⟩2. QED
BY ⟨1⟩1, ⟨3⟩1
⟨2⟩4. QED
BY ⟨2⟩1, ⟨2⟩2, ⟨2⟩3, *ServiceTypeSafe*DEF *sResult*
⟨1⟩8. $\land$ $sResult.newPublicState \in PublicStateType$
$\land$ $sResult.newPrivateState \in PrivateStateType$

$\wedge$ $sResult.output \in OutputType$
BY $\langle 1 \rangle 7$ DEF $ServiceResultType$
$\langle 1 \rangle 9$. $newPrivateStateEnc \in PrivateStateEncType$
$\langle 2 \rangle 1$. $LL2NVRAM.symmetricKey \in SymmetricKeyType$
BY $\langle 1 \rangle 2$, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$
$\langle 2 \rangle 2$. $sResult.newPrivateState \in PrivateStateType$
BY $\langle 1 \rangle 8$
$\langle 2 \rangle 3$. QED
BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, $SymmetricEncryptionTypeSafe$DEF $newPrivateStateEnc$
$\langle 1 \rangle 10$. $currentHistorySummary \in HistorySummaryType$
$\langle 2 \rangle 1$. $LL2NVRAM.historySummaryAnchor \in HashType$
BY $\langle 1 \rangle 2$, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$
$\langle 2 \rangle 2$. $LL2SPCR \in HashType$
BY $\langle 1 \rangle 2$ DEF $LL2TypeInvariant$
$\langle 2 \rangle 3$. QED
BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$ DEF $currentHistorySummary$, $HistorySummaryType$
$\langle 1 \rangle 11$. $\wedge$ $currentHistorySummary.anchor \in HashType$
$\wedge$ $currentHistorySummary.extension \in HashType$
BY $\langle 1 \rangle 10$ DEF $HistorySummaryType$
$\langle 1 \rangle 12$. $newHistorySummary \in HistorySummaryType$
$\langle 2 \rangle 1$. $currentHistorySummary \in HistorySummaryType$
BY $\langle 1 \rangle 10$
$\langle 2 \rangle 2$. $input \in InputType$
$\langle 3 \rangle 1$. $input \in LL2AvailableInputs$
BY $\langle 1 \rangle 1$
$\langle 3 \rangle 2$. $LL2AvailableInputs \subseteq InputType$
BY $\langle 1 \rangle 2$ DEF $LL2TypeInvariant$
$\langle 3 \rangle 3$. QED
BY $\langle 3 \rangle 1$, $\langle 3 \rangle 2$
$\langle 2 \rangle 3$. $LL2NVRAM.hashBarrier \in HashType$
BY $\langle 1 \rangle 2$, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$
$\langle 2 \rangle 4$. QED
BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, $\langle 2 \rangle 3$, $SuccessorTypeSafe$DEF $newHistorySummary$
$\langle 1 \rangle 13$. $\wedge$ $newHistorySummary.anchor \in HashType$
$\wedge$ $newHistorySummary.extension \in HashType$
BY $\langle 1 \rangle 12$ DEF $HistorySummaryType$
$\langle 1 \rangle 14$. $newHistorySummaryHash \in HashType$
$\langle 2 \rangle 1$. $newHistorySummary.anchor \in HashDomain$
$\langle 3 \rangle 1$. $newHistorySummary.anchor \in HashType$
BY $\langle 1 \rangle 12$ DEF $HistorySummaryType$
$\langle 3 \rangle 2$. QED
BY $\langle 3 \rangle 1$ DEF $HashDomain$
$\langle 2 \rangle 2$. $newHistorySummary.extension \in HashDomain$
$\langle 3 \rangle 1$. $newHistorySummary.extension \in HashType$
BY $\langle 1 \rangle 12$ DEF $HistorySummaryType$
$\langle 3 \rangle 2$. QED
BY $\langle 3 \rangle 1$ DEF $HashDomain$
$\langle 2 \rangle 3$. QED
BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, $HashTypeSafe$DEF $newHistorySummaryHash$
$\langle 1 \rangle 15$. $newStateHash \in HashType$
$\langle 2 \rangle 1$. $sResult.newPublicState \in HashDomain$
$\langle 3 \rangle 1$. $sResult.newPublicState \in PublicStateType$

BY $\langle 1 \rangle 8$
$\langle 3 \rangle 2.$ QED
BY $\langle 3 \rangle 1$ DEF $HashDomain$
$\langle 2 \rangle 2.$ $newPrivateStateEnc \in HashDomain$
BY $\langle 1 \rangle 9$ DEF $HashDomain$
$\langle 2 \rangle 3.$ QED
BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, $HashTypeSafe$DEF $newStateHash$
$\langle 1 \rangle 16.$ $newHistoryStateBinding \in HashType$
$\langle 2 \rangle 1.$ $newHistorySummaryHash \in HashDomain$
BY $\langle 1 \rangle 14$ DEF $HashDomain$
$\langle 2 \rangle 2.$ $newStateHash \in HashDomain$
BY $\langle 1 \rangle 15$ DEF $HashDomain$
$\langle 2 \rangle 3.$ QED
BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, $HashTypeSafe$DEF $newHistoryStateBinding$
$\langle 1 \rangle 17.$ $newAuthenticator \in MACType$
$\langle 2 \rangle 1.$ $LL2NVRAM.symmetricKey \in SymmetricKeyType$
BY $\langle 1 \rangle 2$, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$
$\langle 2 \rangle 2.$ $newHistoryStateBinding \in HashType$
BY $\langle 1 \rangle 16$
$\langle 2 \rangle 3.$ QED
BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, $GenerateMACTypeSafe$DEF $newAuthenticator$
$\langle 1 \rangle 18.$ QED
BY $\langle 1 \rangle 3$, $\langle 1 \rangle 4$, $\langle 1 \rangle 5$, $\langle 1 \rangle 6$, $\langle 1 \rangle 7$, $\langle 1 \rangle 8$, $\langle 1 \rangle 9$,
$\langle 1 \rangle 10$, $\langle 1 \rangle 11$, $\langle 1 \rangle 12$, $\langle 1 \rangle 13$, $\langle 1 \rangle 14$, $\langle 1 \rangle 15$, $\langle 1 \rangle 16$, $\langle 1 \rangle 17$
DEF $historySummaryHash$, $stateHash$, $historyStateBinding$, $privateState$,
$sResult$, $newPrivateStateEnc$, $currentHistorySummary$, $newHistorySummary$,
$newHistorySummaryHash$, $newStateHash$, $newHistoryStateBinding$, $newAuthenticator$

The $LL2RepeatOperationDefsTypeSafeLemma$ proves that the definitions within the LET of the $LL2RepeatOperation$ action all have the appropriate type. This is a trivial proof that merely walks through the definitions.

THEOREM $LL2RepeatOperationDefsTypeSafeLemma \triangleq$
$\forall input \in LL2AvailableInputs :$
$LL2TypeInvariant \Rightarrow$
LET
$historySummaryHash \triangleq$
$Hash(LL2RAM.historySummary.anchor, LL2RAM.historySummary.extension)$
$stateHash \triangleq Hash(LL2RAM.publicState, LL2RAM.privateStateEnc)$
$historyStateBinding \triangleq Hash(historySummaryHash, stateHash)$
$newHistorySummary \triangleq$
$Successor(LL2RAM.historySummary, input, LL2NVRAM.hashBarrier)$
$checkpointedHistorySummary \triangleq Checkpoint(LL2RAM.historySummary)$
$newCheckpointedHistorySummary \triangleq$
$Successor(checkpointedHistorySummary, input, LL2NVRAM.hashBarrier)$
$checkpointedNewHistorySummary \triangleq Checkpoint(newHistorySummary)$
$checkpointedNewCheckpointedHistorySummary \triangleq$
$Checkpoint(newCheckpointedHistorySummary)$
$privateState \triangleq SymmetricDecrypt(LL2NVRAM.symmetricKey, LL2RAM.privateStateEnc)$
$sResult \triangleq Service(LL2RAM.publicState, privateState, input)$
$newPrivateStateEnc \triangleq$
$SymmetricEncrypt(LL2NVRAM.symmetricKey, sResult.newPrivateState)$
$currentHistorySummary \triangleq [$
$anchor \mapsto LL2NVRAM.historySummaryAnchor,$

$\quad\quad \mathit{extension} \mapsto \mathit{LL2SPCR}]$
$\quad \mathit{currentHistorySummaryHash} \triangleq \mathit{Hash}(\mathit{LL2NVRAM}.\mathit{historySummaryAnchor}, \mathit{LL2SPCR})$
$\quad \mathit{newStateHash} \triangleq \mathit{Hash}(\mathit{sResult}.\mathit{newPublicState}, \mathit{newPrivateStateEnc})$
$\quad \mathit{newHistoryStateBinding} \triangleq \mathit{Hash}(\mathit{currentHistorySummaryHash}, \mathit{newStateHash})$
$\quad \mathit{newAuthenticator} \triangleq \mathit{GenerateMAC}(\mathit{LL2NVRAM}.\mathit{symmetricKey}, \mathit{newHistoryStateBinding})$
IN
$\land\ \mathit{historySummaryHash} \in \mathit{HashType}$
$\land\ \mathit{stateHash} \in \mathit{HashType}$
$\land\ \mathit{historyStateBinding} \in \mathit{HashType}$
$\land\ \mathit{newHistorySummary} \in \mathit{HistorySummaryType}$
$\land\ \mathit{newHistorySummary}.\mathit{anchor} \in \mathit{HashType}$
$\land\ \mathit{newHistorySummary}.\mathit{extension} \in \mathit{HashType}$
$\land\ \mathit{checkpointedHistorySummary} \in \mathit{HistorySummaryType}$
$\land\ \mathit{checkpointedHistorySummary}.\mathit{anchor} \in \mathit{HashType}$
$\land\ \mathit{checkpointedHistorySummary}.\mathit{extension} \in \mathit{HashType}$
$\land\ \mathit{newCheckpointedHistorySummary} \in \mathit{HistorySummaryType}$
$\land\ \mathit{newCheckpointedHistorySummary}.\mathit{anchor} \in \mathit{HashType}$
$\land\ \mathit{newCheckpointedHistorySummary}.\mathit{extension} \in \mathit{HashType}$
$\land\ \mathit{checkpointedNewHistorySummary} \in \mathit{HistorySummaryType}$
$\land\ \mathit{checkpointedNewHistorySummary}.\mathit{anchor} \in \mathit{HashType}$
$\land\ \mathit{checkpointedNewHistorySummary}.\mathit{extension} \in \mathit{HashType}$
$\land\ \mathit{checkpointedNewCheckpointedHistorySummary} \in \mathit{HistorySummaryType}$
$\land\ \mathit{checkpointedNewCheckpointedHistorySummary}.\mathit{anchor} \in \mathit{HashType}$
$\land\ \mathit{checkpointedNewCheckpointedHistorySummary}.\mathit{extension} \in \mathit{HashType}$
$\land\ \mathit{privateState} \in \mathit{PrivateStateType}$
$\land\ \mathit{sResult} \in \mathit{ServiceResultType}$
$\land\ \mathit{sResult}.\mathit{newPublicState} \in \mathit{PublicStateType}$
$\land\ \mathit{sResult}.\mathit{newPrivateState} \in \mathit{PrivateStateType}$
$\land\ \mathit{sResult}.\mathit{output} \in \mathit{OutputType}$
$\land\ \mathit{newPrivateStateEnc} \in \mathit{PrivateStateEncType}$
$\land\ \mathit{currentHistorySummary} \in \mathit{HistorySummaryType}$
$\land\ \mathit{currentHistorySummary}.\mathit{anchor} \in \mathit{HashType}$
$\land\ \mathit{currentHistorySummary}.\mathit{extension} \in \mathit{HashType}$
$\land\ \mathit{currentHistorySummaryHash} \in \mathit{HashType}$
$\land\ \mathit{newStateHash} \in \mathit{HashType}$
$\land\ \mathit{newHistoryStateBinding} \in \mathit{HashType}$
$\land\ \mathit{newAuthenticator} \in \mathit{MACType}$

$\langle 1 \rangle 1.$ TAKE $\mathit{input} \in \mathit{LL2AvailableInputs}$
$\langle 1 \rangle\ \mathit{historySummaryHash} \triangleq$
$\quad \mathit{Hash}(\mathit{LL2RAM}.\mathit{historySummary}.\mathit{anchor}, \mathit{LL2RAM}.\mathit{historySummary}.\mathit{extension})$
$\langle 1 \rangle\ \mathit{stateHash} \triangleq \mathit{Hash}(\mathit{LL2RAM}.\mathit{publicState}, \mathit{LL2RAM}.\mathit{privateStateEnc})$
$\langle 1 \rangle\ \mathit{historyStateBinding} \triangleq \mathit{Hash}(\mathit{historySummaryHash}, \mathit{stateHash})$
$\langle 1 \rangle\ \mathit{newHistorySummary} \triangleq \mathit{Successor}(\mathit{LL2RAM}.\mathit{historySummary}, \mathit{input}, \mathit{LL2NVRAM}.\mathit{hashBarrier})$
$\langle 1 \rangle\ \mathit{checkpointedHistorySummary} \triangleq \mathit{Checkpoint}(\mathit{LL2RAM}.\mathit{historySummary})$
$\langle 1 \rangle\ \mathit{newCheckpointedHistorySummary} \triangleq$
$\quad \mathit{Successor}(\mathit{checkpointedHistorySummary}, \mathit{input}, \mathit{LL2NVRAM}.\mathit{hashBarrier})$
$\langle 1 \rangle\ \mathit{checkpointedNewHistorySummary} \triangleq \mathit{Checkpoint}(\mathit{newHistorySummary})$
$\langle 1 \rangle\ \mathit{checkpointedNewCheckpointedHistorySummary} \triangleq$
$\quad \mathit{Checkpoint}(\mathit{newCheckpointedHistorySummary})$
$\langle 1 \rangle\ \mathit{privateState} \triangleq \mathit{SymmetricDecrypt}(\mathit{LL2NVRAM}.\mathit{symmetricKey}, \mathit{LL2RAM}.\mathit{privateStateEnc})$
$\langle 1 \rangle\ \mathit{sResult} \triangleq \mathit{Service}(\mathit{LL2RAM}.\mathit{publicState}, \mathit{privateState}, \mathit{input})$
$\langle 1 \rangle\ \mathit{newPrivateStateEnc} \triangleq$

SymmetricEncrypt(*LL2NVRAM.symmetricKey*, *sResult.newPrivateState*)

$\langle 1 \rangle$ *currentHistorySummary* $\triangleq$ [
anchor $\mapsto$ *LL2NVRAM.historySummaryAnchor*,
extension $\mapsto$ *LL2SPCR*]

$\langle 1 \rangle$ *currentHistorySummaryHash* $\triangleq$ *Hash*(*LL2NVRAM.historySummaryAnchor*, *LL2SPCR*)

$\langle 1 \rangle$ *newStateHash* $\triangleq$ *Hash*(*sResult.newPublicState*, *newPrivateStateEnc*)

$\langle 1 \rangle$ *newHistoryStateBinding* $\triangleq$ *Hash*(*currentHistorySummaryHash*, *newStateHash*)

$\langle 1 \rangle$ *newAuthenticator* $\triangleq$ *GenerateMAC*(*LL2NVRAM.symmetricKey*, *newHistoryStateBinding*)

$\langle 1 \rangle$ HIDE DEF *historySummaryHash*, *stateHash*, *historyStateBinding*, *newHistorySummary*, *checkpointedHistorySummary*, *newCheckpointedHistorySummary*, *checkpointedNewHistorySummary*, *checkpointedNewCheckpointedHistorySummary*, *privateState*, *sResult*, *newPrivateStateEnc*, *currentHistorySummary*, *currentHistorySummaryHash*, *newStateHash*, *newHistoryStateBinding*, *newAuthenticator*

$\langle 1 \rangle 2.$ HAVE *LL2TypeInvariant*

$\langle 1 \rangle 3.$ *input* $\in$ *InputType*

  $\langle 2 \rangle 1.$ *input* $\in$ *LL2AvailableInputs*

    BY $\langle 1 \rangle 1$

  $\langle 2 \rangle 2.$ *LL2AvailableInputs* $\subseteq$ *InputType*

    BY $\langle 1 \rangle 2$ DEF *LL2TypeInvariant*

  $\langle 2 \rangle 3.$ QED

    BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$

$\langle 1 \rangle 4.$ *LL2NVRAM.hashBarrier* $\in$ *HashType*

  BY $\langle 1 \rangle 2$, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*

$\langle 1 \rangle 5.$ *historySummaryHash* $\in$ *HashType*

  $\langle 2 \rangle 1.$ *LL2RAM.historySummary.anchor* $\in$ *HashDomain*

    $\langle 3 \rangle 1.$ *LL2RAM.historySummary.anchor* $\in$ *HashType*

      BY $\langle 1 \rangle 2$, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*

    $\langle 3 \rangle 2.$ QED

      BY $\langle 3 \rangle 1$ DEF *HashDomain*

  $\langle 2 \rangle 2.$ *LL2RAM.historySummary.extension* $\in$ *HashDomain*

    $\langle 3 \rangle 1.$ *LL2RAM.historySummary.extension* $\in$ *HashType*

      BY $\langle 1 \rangle 2$, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*

    $\langle 3 \rangle 2.$ QED

      BY $\langle 3 \rangle 1$ DEF *HashDomain*

  $\langle 2 \rangle 3.$ QED

    BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, *HashTypeSafe*DEF *historySummaryHash*

$\langle 1 \rangle 6.$ *stateHash* $\in$ *HashType*

  $\langle 2 \rangle 1.$ $\land$ *LL2RAM.publicState* $\in$ *PublicStateType*
  $\land$ *LL2RAM.privateStateEnc* $\in$ *PrivateStateEncType*

    BY $\langle 1 \rangle 2$, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*

  $\langle 2 \rangle 2.$ $\land$ *LL2RAM.publicState* $\in$ *HashDomain*
  $\land$ *LL2RAM.privateStateEnc* $\in$ *HashDomain*

    BY $\langle 2 \rangle 1$ DEF *HashDomain*

  $\langle 2 \rangle 3.$ QED

    BY $\langle 2 \rangle 2$, *HashTypeSafe*DEF *stateHash*

$\langle 1 \rangle 7.$ *historyStateBinding* $\in$ *HashType*

  $\langle 2 \rangle 1.$ *historySummaryHash* $\in$ *HashDomain*

    BY $\langle 1 \rangle 5$ DEF *HashDomain*

  $\langle 2 \rangle 2.$ *stateHash* $\in$ *HashDomain*

    BY $\langle 1 \rangle 6$ DEF *HashDomain*

  $\langle 2 \rangle 3.$ QED

BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, *HashTypeSafe* DEF *historyStateBinding*
$\langle 1 \rangle 8$. *newHistorySummary* $\in$ *HistorySummaryType*
$\langle 2 \rangle 1$. *LL2RAM.historySummary* $\in$ *HistorySummaryType*
BY $\langle 1 \rangle 2$, *LL2SubtypeImplicationLemma* DEF *LL2SubtypeImplication*
$\langle 2 \rangle 2$. QED
BY $\langle 1 \rangle 3$, $\langle 1 \rangle 4$, $\langle 2 \rangle 1$, *SuccessorTypeSafe* DEF *newHistorySummary*
$\langle 1 \rangle 9$. $\wedge$ *newHistorySummary.anchor* $\in$ *HashType*
$\wedge$ *newHistorySummary.extension* $\in$ *HashType*
BY $\langle 1 \rangle 8$ DEF *HistorySummaryType*
$\langle 1 \rangle 10$. *checkpointedHistorySummary* $\in$ *HistorySummaryType*
$\langle 2 \rangle 1$. *LL2RAM.historySummary* $\in$ *HistorySummaryType*
BY $\langle 1 \rangle 2$, *LL2SubtypeImplicationLemma* DEF *LL2SubtypeImplication*
$\langle 2 \rangle 2$. QED
BY $\langle 2 \rangle 1$, *CheckpointTypeSafe* DEF *checkpointedHistorySummary*
$\langle 1 \rangle 11$. $\wedge$ *checkpointedHistorySummary.anchor* $\in$ *HashType*
$\wedge$ *checkpointedHistorySummary.extension* $\in$ *HashType*
BY $\langle 1 \rangle 10$ DEF *HistorySummaryType*
$\langle 1 \rangle 12$. *newCheckpointedHistorySummary* $\in$ *HistorySummaryType*
BY $\langle 1 \rangle 3$, $\langle 1 \rangle 4$, $\langle 1 \rangle 10$, *SuccessorTypeSafe* DEF *newCheckpointedHistorySummary*
$\langle 1 \rangle 13$. $\wedge$ *newCheckpointedHistorySummary.anchor* $\in$ *HashType*
$\wedge$ *newCheckpointedHistorySummary.extension* $\in$ *HashType*
BY $\langle 1 \rangle 12$ DEF *HistorySummaryType*
$\langle 1 \rangle 14$. *checkpointedNewHistorySummary* $\in$ *HistorySummaryType*
BY $\langle 1 \rangle 8$, *CheckpointTypeSafe* DEF *checkpointedNewHistorySummary*
$\langle 1 \rangle 15$. $\wedge$ *checkpointedNewHistorySummary.anchor* $\in$ *HashType*
$\wedge$ *checkpointedNewHistorySummary.extension* $\in$ *HashType*
BY $\langle 1 \rangle 14$ DEF *HistorySummaryType*
$\langle 1 \rangle 16$. *checkpointedNewCheckpointedHistorySummary* $\in$ *HistorySummaryType*
BY $\langle 1 \rangle 12$, *CheckpointTypeSafe* DEF *checkpointedNewCheckpointedHistorySummary*
$\langle 1 \rangle 17$. $\wedge$ *checkpointedNewCheckpointedHistorySummary.anchor* $\in$ *HashType*
$\wedge$ *checkpointedNewCheckpointedHistorySummary.extension* $\in$ *HashType*
BY $\langle 1 \rangle 16$ DEF *HistorySummaryType*
$\langle 1 \rangle 18$. *privateState* $\in$ *PrivateStateType*
$\langle 2 \rangle 1$. $\wedge$ *LL2NVRAM.symmetricKey* $\in$ *SymmetricKeyType*
$\wedge$ *LL2RAM.privateStateEnc* $\in$ *PrivateStateEncType*
BY $\langle 1 \rangle 2$, *LL2SubtypeImplicationLemma* DEF *LL2SubtypeImplication*
$\langle 2 \rangle 2$. QED
BY $\langle 2 \rangle 1$, *SymmetricDecryptionTypeSafe* DEF *privateState*
$\langle 1 \rangle 19$. *sResult* $\in$ *ServiceResultType*
$\langle 2 \rangle 1$. *LL2RAM.publicState* $\in$ *PublicStateType*
BY $\langle 1 \rangle 2$, *LL2SubtypeImplicationLemma* DEF *LL2SubtypeImplication*
$\langle 2 \rangle 2$. *privateState* $\in$ *PrivateStateType*
BY $\langle 1 \rangle 18$
$\langle 2 \rangle 3$. *input* $\in$ *InputType*
$\langle 3 \rangle 1$. *LL2AvailableInputs* $\subseteq$ *InputType*
BY $\langle 1 \rangle 2$ DEF *LL2TypeInvariant*
$\langle 3 \rangle 2$. QED
BY $\langle 1 \rangle 1$, $\langle 3 \rangle 1$
$\langle 2 \rangle 4$. QED
BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, $\langle 2 \rangle 3$, *ServiceTypeSafe* DEF *sResult*
$\langle 1 \rangle 20$. $\wedge$ *sResult.newPublicState* $\in$ *PublicStateType*
$\wedge$ *sResult.newPrivateState* $\in$ *PrivateStateType*

$\wedge$ $sResult.output \in OutputType$

BY $\langle 1 \rangle 19$ DEF $ServiceResultType$

$\langle 1 \rangle 21$. $newPrivateStateEnc \in PrivateStateEncType$

$\langle 2 \rangle 1$. $LL2NVRAM.symmetricKey \in SymmetricKeyType$

BY $\langle 1 \rangle 2$, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$

$\langle 2 \rangle 2$. $sResult.newPrivateState \in PrivateStateType$

BY $\langle 1 \rangle 20$

$\langle 2 \rangle 3$. QED

BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, $SymmetricEncryptionTypeSafe$DEF $newPrivateStateEnc$

$\langle 1 \rangle 22$. $currentHistorySummary \in HistorySummaryType$

$\langle 2 \rangle 1$. $LL2NVRAM.historySummaryAnchor \in HashType$

BY $\langle 1 \rangle 2$, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$

$\langle 2 \rangle 2$. $LL2SPCR \in HashType$

BY $\langle 1 \rangle 2$ DEF $LL2TypeInvariant$

$\langle 2 \rangle 3$. QED

BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$ DEF $currentHistorySummary$, $HistorySummaryType$

$\langle 1 \rangle 23$. $\wedge$ $currentHistorySummary.anchor \in HashType$

$\wedge$ $currentHistorySummary.extension \in HashType$

BY $\langle 1 \rangle 22$ DEF $HistorySummaryType$

$\langle 1 \rangle 24$. $currentHistorySummaryHash \in HashType$

$\langle 2 \rangle 1$. $LL2NVRAM.historySummaryAnchor \in HashDomain$

$\langle 3 \rangle 1$. $LL2NVRAM.historySummaryAnchor \in HashType$

BY $\langle 1 \rangle 2$, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$

$\langle 3 \rangle 2$. QED

BY $\langle 3 \rangle 1$ DEF $HashDomain$

$\langle 2 \rangle 2$. $LL2SPCR \in HashDomain$

$\langle 3 \rangle 1$. $LL2SPCR \in HashType$

BY $\langle 1 \rangle 2$ DEF $LL2TypeInvariant$

$\langle 3 \rangle 2$. QED

BY $\langle 3 \rangle 1$ DEF $HashDomain$

$\langle 2 \rangle 3$. QED

BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, $HashTypeSafe$DEF $currentHistorySummaryHash$

$\langle 1 \rangle 25$. $newStateHash \in HashType$

$\langle 2 \rangle 1$. $sResult.newPublicState \in HashDomain$

$\langle 3 \rangle 1$. $sResult.newPublicState \in PublicStateType$

BY $\langle 1 \rangle 20$

$\langle 3 \rangle 2$. QED

BY $\langle 3 \rangle 1$ DEF $HashDomain$

$\langle 2 \rangle 2$. $newPrivateStateEnc \in HashDomain$

BY $\langle 1 \rangle 21$ DEF $HashDomain$

$\langle 2 \rangle 3$. QED

BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, $HashTypeSafe$DEF $newStateHash$

$\langle 1 \rangle 26$. $newHistoryStateBinding \in HashType$

$\langle 2 \rangle 1$. $currentHistorySummaryHash \in HashDomain$

BY $\langle 1 \rangle 24$ DEF $HashDomain$

$\langle 2 \rangle 2$. $newStateHash \in HashDomain$

BY $\langle 1 \rangle 25$ DEF $HashDomain$

$\langle 2 \rangle 3$. QED

BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, $HashTypeSafe$DEF $newHistoryStateBinding$

$\langle 1 \rangle 27$. $newAuthenticator \in MACType$

$\langle 2 \rangle 1$. $LL2NVRAM.symmetricKey \in SymmetricKeyType$

BY $\langle 1 \rangle 2$, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$

⟨2⟩2. *newHistoryStateBinding* ∈ *HashType*
  BY ⟨1⟩26
⟨2⟩3. QED
  BY ⟨2⟩1, ⟨2⟩2, *GenerateMACTypeSafe* DEF *newAuthenticator*
⟨1⟩28. QED
  BY ⟨1⟩5, ⟨1⟩6, ⟨1⟩7, ⟨1⟩8, ⟨1⟩9, ⟨1⟩10, ⟨1⟩11, ⟨1⟩12, ⟨1⟩13, ⟨1⟩14, ⟨1⟩15,
    ⟨1⟩16, ⟨1⟩17, ⟨1⟩18, ⟨1⟩19, ⟨1⟩20, ⟨1⟩21, ⟨1⟩22, ⟨1⟩23, ⟨1⟩24, ⟨1⟩25, ⟨1⟩26, ⟨1⟩27
  DEF *historySummaryHash*, *stateHash*, *historyStateBinding*, *newHistorySummary*,
    checkpointedHistorySummary, *newCheckpointedHistorySummary*,
    checkpointedNewHistorySummary, *checkpointedNewCheckpointedHistorySummary*,
    privateState, *sResult*, *newPrivateStateEnc*, *currentHistorySummary*,
    currentHistorySummaryHash, *newStateHash*, *newHistoryStateBinding*,
    newAuthenticator

The *LL2TakeCheckpointDefsTypeSafeLemma* proves that the definition within the LET of the *LL2TakeCheckpoint* action has the appropriate type. This is a trivial proof that merely walks through the definitions.

THEOREM *LL2TakeCheckpointDefsTypeSafeLemma* ≜
  LL2TypeInvariant ⇒
    LET
      newHistorySummaryAnchor ≜ *Hash*(*LL2NVRAM.historySummaryAnchor*, *LL2SPCR*)
    IN
      newHistorySummaryAnchor ∈ *HashType*
⟨1⟩ *newHistorySummaryAnchor* ≜ *Hash*(*LL2NVRAM.historySummaryAnchor*, *LL2SPCR*)
⟨1⟩ HIDE DEF *newHistorySummaryAnchor*
⟨1⟩1. HAVE *LL2TypeInvariant*
⟨1⟩2. *newHistorySummaryAnchor* ∈ *HashType*
  ⟨2⟩1. *LL2NVRAM.historySummaryAnchor* ∈ *HashDomain*
    ⟨3⟩1. *LL2NVRAM.historySummaryAnchor* ∈ *HashType*
      BY ⟨1⟩1, *LL2SubtypeImplicationLemma* DEF *LL2SubtypeImplication*
    ⟨3⟩2. QED
      BY ⟨3⟩1 DEF *HashDomain*
  ⟨2⟩2. *LL2SPCR* ∈ *HashDomain*
    ⟨3⟩1. *LL2SPCR* ∈ *HashType*
      BY ⟨1⟩1 DEF *LL2TypeInvariant*
    ⟨3⟩2. QED
      BY ⟨3⟩1 DEF *HashDomain*
  ⟨2⟩3. QED
    BY ⟨2⟩1, ⟨2⟩2, *HashTypeSafe* DEF *newHistorySummaryAnchor*
⟨1⟩3. QED
  BY ⟨1⟩2 DEF *newHistorySummaryAnchor*

The *LL2CorruptSPCRDefsTypeSafeLemma* proves that the definition within the LET of the *LL2CorruptSPCR* action has the appropriate type. This is a trivial proof that merely walks through the definitions.

THEOREM *LL2CorruptSPCRDefsTypeSafeLemma* ≜
  ∀ *fakeHash* ∈ *HashDomain* :
    LL2TypeInvariant ⇒
      LET
        newHistorySummaryExtension ≜ *Hash*(*LL2SPCR*, *fakeHash*)

IN

$newHistorySummaryExtension \in HashType$

$\langle 1 \rangle 1.$ TAKE $fakeHash \in HashDomain$

$\langle 1 \rangle$ $newHistorySummaryExtension \triangleq Hash(LL2SPCR, fakeHash)$

$\langle 1 \rangle$ HIDE DEF $newHistorySummaryExtension$

$\langle 1 \rangle 2.$ HAVE $LL2TypeInvariant$

$\langle 1 \rangle 3.$ $newHistorySummaryExtension \in HashType$

$\langle 2 \rangle 1.$ $LL2SPCR \in HashDomain$

$\langle 3 \rangle 1.$ $LL2SPCR \in HashType$

BY $\langle 1 \rangle 2$ DEF $LL2TypeInvariant$

$\langle 3 \rangle 2.$ QED

BY $\langle 3 \rangle 1$ DEF $HashDomain$

$\langle 2 \rangle 2.$ $fakeHash \in HashDomain$

BY $\langle 1 \rangle 1$

$\langle 2 \rangle 3.$ QED

BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, $HashTypeSafe$DEF $newHistorySummaryExtension$

$\langle 1 \rangle 4.$ QED

BY $\langle 1 \rangle 3$ DEF $newHistorySummaryExtension$

The *AuthenticatorsMatchDefsTypeSafeLemma* proves that the definitions within the LET of the *AuthenticatorsMatch* predicate have the appropriate type. This is a trivial proof that merely walks through the definitions.

THEOREM $AuthenticatorsMatchDefsTypeSafeLemma \triangleq$

$\forall$ $stateHash \in HashType$,

$ll1HistorySummary \in HashType$,

$ll2HistorySummary \in HistorySummaryType$:

LET

$ll1HistoryStateBinding \triangleq Hash(ll1HistorySummary, stateHash)$

$ll2HistorySummaryHash \triangleq Hash(ll2HistorySummary.anchor, ll2HistorySummary.extension)$

$ll2HistoryStateBinding \triangleq Hash(ll2HistorySummaryHash, stateHash)$

IN

$\land$ $ll1HistoryStateBinding \in HashType$

$\land$ $ll2HistorySummaryHash \in HashType$

$\land$ $ll2HistoryStateBinding \in HashType$

$\langle 1 \rangle 1.$ TAKE $stateHash \in HashType$,

$ll1HistorySummary \in HashType$,

$ll2HistorySummary \in HistorySummaryType$

$\langle 1 \rangle$ $ll1HistoryStateBinding \triangleq Hash(ll1HistorySummary, stateHash)$

$\langle 1 \rangle$ $ll2HistorySummaryHash \triangleq Hash(ll2HistorySummary.anchor, ll2HistorySummary.extension)$

$\langle 1 \rangle$ $ll2HistoryStateBinding \triangleq Hash(ll2HistorySummaryHash, stateHash)$

$\langle 1 \rangle$ HIDE DEF $ll1HistoryStateBinding$, $ll2HistorySummaryHash$, $ll2HistoryStateBinding$

$\langle 1 \rangle 2.$ $ll1HistoryStateBinding \in HashType$

$\langle 2 \rangle 1.$ $ll1HistorySummary \in HashDomain$

$\langle 3 \rangle 1.$ $ll1HistorySummary \in HashType$

BY $\langle 1 \rangle 1$

$\langle 3 \rangle 2.$ QED

BY $\langle 3 \rangle 1$ DEF $HashDomain$

$\langle 2 \rangle 2.$ $stateHash \in HashDomain$

$\langle 3 \rangle 1.$ $stateHash \in HashType$

BY $\langle 1 \rangle 1$

$\langle 3 \rangle 2.$ QED

BY $\langle 3\rangle 1$ DEF *HashDomain*
$\langle 2\rangle 3.$ QED
BY $\langle 2\rangle 1$, $\langle 2\rangle 2$, *HashTypeSafe*DEF *ll1HistoryStateBinding*
$\langle 1\rangle 3.$ $ll2HistorySummaryHash \in HashType$
$\langle 2\rangle 1.$ $ll2HistorySummary \in HistorySummaryType$
BY $\langle 1\rangle 1$
$\langle 2\rangle 2.$ $ll2HistorySummary.anchor \in HashDomain$
$\langle 3\rangle 1.$ $ll2HistorySummary.anchor \in HashType$
BY $\langle 2\rangle 1$ DEF *HistorySummaryType*
$\langle 3\rangle 2.$ QED
BY $\langle 3\rangle 1$ DEF *HashDomain*
$\langle 2\rangle 3.$ $ll2HistorySummary.extension \in HashDomain$
$\langle 3\rangle 1.$ $ll2HistorySummary.extension \in HashType$
BY $\langle 2\rangle 1$ DEF *HistorySummaryType*
$\langle 3\rangle 2.$ QED
BY $\langle 3\rangle 1$ DEF *HashDomain*
$\langle 2\rangle 4.$ QED
BY $\langle 2\rangle 2$, $\langle 2\rangle 3$, *HashTypeSafe*DEF *ll2HistorySummaryHash*
$\langle 1\rangle 4.$ $ll2HistoryStateBinding \in HashType$
$\langle 2\rangle 1.$ $ll2HistorySummaryHash \in HashDomain$
$\langle 3\rangle 1.$ $ll2HistorySummaryHash \in HashType$
BY $\langle 1\rangle 3$
$\langle 3\rangle 2.$ QED
BY $\langle 3\rangle 1$ DEF *HashDomain*
$\langle 2\rangle 2.$ $stateHash \in HashDomain$
$\langle 3\rangle 1.$ $stateHash \in HashType$
BY $\langle 1\rangle 1$
$\langle 3\rangle 2.$ QED
BY $\langle 3\rangle 1$ DEF *HashDomain*
$\langle 2\rangle 3.$ QED
BY $\langle 2\rangle 1$, $\langle 2\rangle 2$, *HashTypeSafe*DEF *ll2HistoryStateBinding*
$\langle 1\rangle 5.$ QED
BY $\langle 1\rangle 2$, $\langle 1\rangle 3$, $\langle 1\rangle 4$
DEF *ll1HistoryStateBinding*, *ll2HistorySummaryHash*, *ll2HistoryStateBinding*

The *TypeSafetyRefinementLemma* states that if the Memoir-Opt type invariant holds, and the refinement holds, then the Memoir-Basic type invariant holds.

THEOREM *TypeSafetyRefinementLemma* $\triangleq$
$LL2TypeInvariant \land LL2Refinement \Rightarrow LL1TypeInvariant$
$\langle 1\rangle 1.$ HAVE $LL2TypeInvariant \land LL2Refinement$
$\langle 1\rangle 2.$ $LL1AvailableInputs \subseteq InputType$
$\langle 2\rangle 1.$ $LL1AvailableInputs = LL2AvailableInputs$
BY $\langle 1\rangle 1$ DEF *LL2Refinement*
$\langle 2\rangle 2.$ $LL2AvailableInputs \subseteq InputType$
BY $\langle 1\rangle 1$ DEF *LL2TypeInvariant*
$\langle 2\rangle 3.$ QED
BY $\langle 2\rangle 1$, $\langle 2\rangle 2$
$\langle 1\rangle 3.$ $LL1ObservedOutputs \subseteq OutputType$
$\langle 2\rangle 1.$ $LL1ObservedOutputs = LL2ObservedOutputs$
BY $\langle 1\rangle 1$ DEF *LL2Refinement*

$\langle 2 \rangle 2.$ $LL2ObservedOutputs \subseteq OutputType$
BY $\langle 1 \rangle 1$ DEF $LL2TypeInvariant$
$\langle 2 \rangle 3.$ QED
BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$
$\langle 1 \rangle 4.$ $LL1ObservedAuthenticators \subseteq MACType$
BY $\langle 1 \rangle 1$ DEF $LL2Refinement$
$\langle 1 \rangle 5.$ $LL1Disk \in LL1UntrustedStorageType$
BY $\langle 1 \rangle 1$ DEF $LL2Refinement$
$\langle 1 \rangle 6.$ $LL1RAM \in LL1UntrustedStorageType$
BY $\langle 1 \rangle 1$ DEF $LL2Refinement$
$\langle 1 \rangle 7.$ $LL1NVRAM \in LL1TrustedStorageType$
BY $\langle 1 \rangle 1$ DEF $LL2Refinement$
$\langle 1 \rangle 8.$ QED
BY $\langle 1 \rangle 2$, $\langle 1 \rangle 3$, $\langle 1 \rangle 4$, $\langle 1 \rangle 5$, $\langle 1 \rangle 6$, $\langle 1 \rangle 7$ DEF $LL1TypeInvariant$

4.9 Proof of Type Safety of the Memoir-Opt Spec

MODULE *MemoirLL2TypeSafety*

This module proves the type safety of the Memoir-Opt spec.

EXTENDS *MemoirLL2TypeLemmas*

THEOREM $LL2TypeSafe \triangleq LL2Spec \Rightarrow \Box LL2TypeInvariant$

The top level of the proof is boilerplate TLA+ for an *Inv*1-style proof. First, we prove that the initial state satisfies *LL2TypeInvariant*. Second, we prove that the *LL2Next* predicate inductively preserves *LL2TypeInvariant*. Third, we use temporal induction to prove that these two conditions satisfy type safety over all behaviors.

$\langle 1 \rangle 1.\ LL2Init \Rightarrow LL2TypeInvariant$

The base case follows directly from the definition of *LL2Init*. There are a bunch of steps, but they are simple expansions of definitions and appeals to the type safety of the initial definitions.

$\langle 2 \rangle 1.$ HAVE $LL2Init$

$\langle 2 \rangle 2.$ PICK $symmetricKey \in SymmetricKeyType,\ hashBarrier \in HashType :$
$\quad LL2Init!(symmetricKey, hashBarrier)!1$

BY $\langle 2 \rangle 1$ DEF $LL2Init$

$\langle 2 \rangle\ initialPrivateStateEnc \triangleq SymmetricEncrypt(symmetricKey, InitialPrivateState)$

$\langle 2 \rangle\ initialStateHash \triangleq Hash(InitialPublicState, initialPrivateStateEnc)$

$\langle 2 \rangle\ initialHistorySummary \triangleq [$
$\quad anchor \mapsto BaseHashValue,$
$\quad extension \mapsto BaseHashValue]$

$\langle 2 \rangle\ initialHistorySummaryHash \triangleq Hash(BaseHashValue, BaseHashValue)$

$\langle 2 \rangle\ initialHistoryStateBinding \triangleq Hash(initialHistorySummaryHash, initialStateHash)$

$\langle 2 \rangle\ initialAuthenticator \triangleq GenerateMAC(symmetricKey, initialHistoryStateBinding)$

$\langle 2 \rangle\ initialUntrustedStorage \triangleq [$
$\quad publicState \mapsto InitialPublicState,$
$\quad privateStateEnc \mapsto initialPrivateStateEnc,$
$\quad historySummary \mapsto initialHistorySummary,$
$\quad authenticator \mapsto initialAuthenticator]$

$\langle 2 \rangle\ initialTrustedStorage \triangleq [$
$\quad historySummaryAnchor \mapsto BaseHashValue,$
$\quad symmetricKey \mapsto symmetricKey,$
$\quad hashBarrier \mapsto hashBarrier,$
$\quad extensionInProgress \mapsto \text{FALSE}]$

$\langle 2 \rangle 3.$
$\quad \land\ initialPrivateStateEnc \in PrivateStateEncType$
$\quad \land\ initialStateHash \in HashType$
$\quad \land\ initialHistorySummary \in HistorySummaryType$
$\quad \land\ initialHistorySummaryHash \in HashType$
$\quad \land\ initialHistoryStateBinding \in HashType$
$\quad \land\ initialAuthenticator \in MACType$
$\quad \land\ initialUntrustedStorage \in LL2UntrustedStorageType$
$\quad \land\ initialTrustedStorage \in LL2TrustedStorageType$

$\langle 3 \rangle 1.\ symmetricKey \in SymmetricKeyType$

BY $\langle 2 \rangle 2$

$\langle 3 \rangle 2.$ QED

BY $\langle 3 \rangle 1$, $LL2InitDefsTypeSafeLemma$

$\langle 2 \rangle$ HIDE DEF $initialPrivateStateEnc, initialStateHash, initialHistorySummary,$
$\quad initialHistorySummaryHash, initialHistoryStateBinding, initialAuthenticator,$
$\quad initialUntrustedStorage, initialTrustedStorage$

$\langle 2 \rangle 4.\ LL2AvailableInputs \subseteq InputType$

$\langle 3 \rangle 1.\ LL2AvailableInputs = InitialAvailableInputs$

BY $\langle 2 \rangle 2$
$\langle 3 \rangle 2$. $InitialAvailableInputs \subseteq InputType$
BY $ConstantsTypeSafe$DEF $ConstantsTypeSafe$
$\langle 3 \rangle 3$. QED
BY $\langle 3 \rangle 1$, $\langle 3 \rangle 2$
$\langle 2 \rangle 5$. $LL2ObservedOutputs \subseteq OutputType$
$\langle 3 \rangle 1$. $LL2ObservedOutputs = \{\}$
BY $\langle 2 \rangle 2$
$\langle 3 \rangle 2$. QED
BY $\langle 3 \rangle 1$
$\langle 2 \rangle 6$. $LL2ObservedAuthenticators \subseteq MACType$
$\langle 3 \rangle 1$. $LL2ObservedAuthenticators = \{initialAuthenticator\}$
BY $\langle 2 \rangle 2$
DEF $initialAuthenticator$, $initialHistoryStateBinding$, $initialHistorySummaryHash$, $initialStateHash$, $initialPrivateStateEnc$
$\langle 3 \rangle 2$. $initialAuthenticator \in MACType$
BY $\langle 2 \rangle 3$
$\langle 3 \rangle 3$. QED
BY $\langle 3 \rangle 1$, $\langle 3 \rangle 2$
$\langle 2 \rangle 7$. $LL2Disk \in LL2UntrustedStorageType$
$\langle 3 \rangle 1$. $LL2Disk = initialUntrustedStorage$
BY $\langle 2 \rangle 2$
DEF $initialUntrustedStorage$, $initialHistorySummary$, $initialAuthenticator$, $initialHistorySummaryHash$, $initialHistoryStateBinding$, $initialStateHash$, $initialPrivateStateEnc$
$\langle 3 \rangle 2$. $initialUntrustedStorage \in LL2UntrustedStorageType$
BY $\langle 2 \rangle 3$
$\langle 3 \rangle 3$. QED
BY $\langle 3 \rangle 1$, $\langle 3 \rangle 2$
$\langle 2 \rangle 8$. $LL2RAM \in LL2UntrustedStorageType$
$\langle 3 \rangle 1$. $LL2RAM = initialUntrustedStorage$
BY $\langle 2 \rangle 2$
DEF $initialUntrustedStorage$, $initialHistorySummary$, $initialAuthenticator$, $initialHistorySummaryHash$, $initialHistoryStateBinding$, $initialStateHash$, $initialPrivateStateEnc$
$\langle 3 \rangle 2$. $initialUntrustedStorage \in LL2UntrustedStorageType$
BY $\langle 2 \rangle 3$
$\langle 3 \rangle 3$. QED
BY $\langle 3 \rangle 1$, $\langle 3 \rangle 2$
$\langle 2 \rangle 9$. $LL2NVRAM \in LL2TrustedStorageType$
$\langle 3 \rangle 1$. $LL2NVRAM = initialTrustedStorage$
BY $\langle 2 \rangle 2$ DEF $initialTrustedStorage$
$\langle 3 \rangle 2$. $initialTrustedStorage \in LL2TrustedStorageType$
BY $\langle 2 \rangle 3$
$\langle 3 \rangle 3$. QED
BY $\langle 3 \rangle 1$, $\langle 3 \rangle 2$
$\langle 2 \rangle 10$. $LL2SPCR \in HashType$
$\langle 3 \rangle 1$. $LL2SPCR = BaseHashValue$
BY $\langle 2 \rangle 2$
$\langle 3 \rangle 2$. $BaseHashValue \in HashType$
BY $BaseHashValueTypeSafe$
$\langle 3 \rangle 3$. QED
BY $\langle 3 \rangle 1$, $\langle 3 \rangle 2$
$\langle 2 \rangle 11$. QED

BY $\langle 2\rangle 4$, $\langle 2\rangle 5$, $\langle 2\rangle 6$, $\langle 2\rangle 7$, $\langle 2\rangle 8$, $\langle 2\rangle 9$, $\langle 2\rangle 10$ DEF $LL2TypeInvariant$

$\langle 1\rangle 2$. $LL2TypeInvariant \land [LL2Next]_{LL2Vars} \Rightarrow LL2TypeInvariant'$

The induction step is also straightforward. We assume the antecedents of the implication, then show that the consequent holds for all nine *LL2Next* actions plus stuttering.

$\langle 2\rangle 1$. HAVE $LL2TypeInvariant \land [LL2Next]_{LL2Vars}$

$\langle 2\rangle 2$. CASE UNCHANGED $LL2Vars$

Type safety is inductively trivial for a stuttering step.

$\langle 3\rangle 1$. $LL2AvailableInputs' \subseteq InputType$
$\langle 4\rangle 1$. $LL2AvailableInputs \subseteq InputType$
BY $\langle 2\rangle 1$ DEF $LL2TypeInvariant$
$\langle 4\rangle 2$. UNCHANGED $LL2AvailableInputs$
BY $\langle 2\rangle 2$ DEF $LL2Vars$
$\langle 4\rangle 3$. QED
BY $\langle 4\rangle 1$, $\langle 4\rangle 2$

$\langle 3\rangle 2$. $LL2ObservedOutputs' \subseteq OutputType$
$\langle 4\rangle 1$. $LL2ObservedOutputs \subseteq OutputType$
BY $\langle 2\rangle 1$ DEF $LL2TypeInvariant$
$\langle 4\rangle 2$. UNCHANGED $LL2ObservedOutputs$
BY $\langle 2\rangle 2$ DEF $LL2Vars$
$\langle 4\rangle 3$. QED
BY $\langle 4\rangle 1$, $\langle 4\rangle 2$

$\langle 3\rangle 3$. $LL2ObservedAuthenticators' \subseteq MACType$
$\langle 4\rangle 1$. $LL2ObservedAuthenticators \subseteq MACType$
BY $\langle 2\rangle 1$ DEF $LL2TypeInvariant$
$\langle 4\rangle 2$. UNCHANGED $LL2ObservedAuthenticators$
BY $\langle 2\rangle 2$ DEF $LL2Vars$
$\langle 4\rangle 3$. QED
BY $\langle 4\rangle 1$, $\langle 4\rangle 2$

$\langle 3\rangle 4$. $LL2Disk' \in LL2UntrustedStorageType$
$\langle 4\rangle 1$. $LL2Disk \in LL2UntrustedStorageType$
BY $\langle 2\rangle 1$ DEF $LL2TypeInvariant$
$\langle 4\rangle 2$. UNCHANGED $LL2Disk$
BY $\langle 2\rangle 2$ DEF $LL2Vars$
$\langle 4\rangle 3$. QED
BY $\langle 4\rangle 1$, $\langle 4\rangle 2$

$\langle 3\rangle 5$. $LL2RAM' \in LL2UntrustedStorageType$
$\langle 4\rangle 1$. $LL2RAM \in LL2UntrustedStorageType$
BY $\langle 2\rangle 1$ DEF $LL2TypeInvariant$
$\langle 4\rangle 2$. UNCHANGED $LL2RAM$
BY $\langle 2\rangle 2$ DEF $LL2Vars$
$\langle 4\rangle 3$. QED
BY $\langle 4\rangle 1$, $\langle 4\rangle 2$

$\langle 3\rangle 6$. $LL2NVRAM' \in LL2TrustedStorageType$
$\langle 4\rangle 1$. $LL2NVRAM \in LL2TrustedStorageType$
BY $\langle 2\rangle 1$ DEF $LL2TypeInvariant$
$\langle 4\rangle 2$. UNCHANGED $LL2NVRAM$
BY $\langle 2\rangle 2$ DEF $LL2Vars$
$\langle 4\rangle 3$. QED
BY $\langle 4\rangle 1$, $\langle 4\rangle 2$

$\langle 3\rangle 7$. $LL2SPCR' \in HashType$
$\langle 4\rangle 1$. $LL2SPCR \in HashType$
BY $\langle 2\rangle 1$ DEF $LL2TypeInvariant$

⟨4⟩2. UNCHANGED $LL2SPCR$
BY ⟨2⟩2 DEF $LL2Vars$
⟨4⟩3. QED
BY ⟨4⟩1, ⟨4⟩2
⟨3⟩8. QED
BY ⟨3⟩1, ⟨3⟩2, ⟨3⟩3, ⟨3⟩4, ⟨3⟩5, ⟨3⟩6, ⟨3⟩7 DEF $LL2TypeInvariant$
⟨2⟩3. CASE $LL2Next$
⟨3⟩1. CASE $LL2MakeInputAvailable$
Type safety is straightforward for a *LL2MakeInputAvailable* action.
⟨4⟩1. PICK $input \in InputType : LL2MakeInputAvailable!(input)$
BY ⟨3⟩1 DEF $LL2MakeInputAvailable$
⟨4⟩2. $LL2AvailableInputs' \subseteq InputType$
⟨5⟩1. $LL2AvailableInputs \subseteq InputType$
BY ⟨2⟩1 DEF $LL2TypeInvariant$
⟨5⟩2. $LL2AvailableInputs' = LL2AvailableInputs \cup \{input\}$
BY ⟨4⟩1
⟨5⟩3. $input \in InputType$
BY ⟨4⟩1
⟨5⟩4. QED
BY ⟨5⟩1, ⟨5⟩2, ⟨5⟩3
⟨4⟩3. $LL2ObservedOutputs' \subseteq OutputType$
⟨5⟩1. $LL2ObservedOutputs \subseteq OutputType$
BY ⟨2⟩1 DEF $LL2TypeInvariant$
⟨5⟩2. UNCHANGED $LL2ObservedOutputs$
BY ⟨4⟩1
⟨5⟩3. QED
BY ⟨5⟩1, ⟨5⟩2
⟨4⟩4. $LL2ObservedAuthenticators' \subseteq MACType$
⟨5⟩1. $LL2ObservedAuthenticators \subseteq MACType$
BY ⟨2⟩1 DEF $LL2TypeInvariant$
⟨5⟩2. UNCHANGED $LL2ObservedAuthenticators$
BY ⟨4⟩1
⟨5⟩3. QED
BY ⟨5⟩1, ⟨5⟩2
⟨4⟩5. $LL2Disk' \in LL2UntrustedStorageType$
⟨5⟩1. $LL2Disk \in LL2UntrustedStorageType$
BY ⟨2⟩1 DEF $LL2TypeInvariant$
⟨5⟩2. UNCHANGED $LL2Disk$
BY ⟨4⟩1
⟨5⟩3. QED
BY ⟨5⟩1, ⟨5⟩2
⟨4⟩6. $LL2RAM' \in LL2UntrustedStorageType$
⟨5⟩1. $LL2RAM \in LL2UntrustedStorageType$
BY ⟨2⟩1 DEF $LL2TypeInvariant$
⟨5⟩2. UNCHANGED $LL2RAM$
BY ⟨4⟩1
⟨5⟩3. QED
BY ⟨5⟩1, ⟨5⟩2
⟨4⟩7. $LL2NVRAM' \in LL2TrustedStorageType$
⟨5⟩1. $LL2NVRAM \in LL2TrustedStorageType$
BY ⟨2⟩1 DEF $LL2TypeInvariant$
⟨5⟩2. UNCHANGED $LL2NVRAM$

BY $\langle 4 \rangle 1$
$\langle 5 \rangle 3.$ QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$
$\langle 4 \rangle 8.$ $LL2SPCR' \in HashType$
$\langle 5 \rangle 1.$ $LL2SPCR \in HashType$
BY $\langle 2 \rangle 1$ DEF $LL2TypeInvariant$
$\langle 5 \rangle 2.$ UNCHANGED $LL2SPCR$
BY $\langle 4 \rangle 1$
$\langle 5 \rangle 3.$ QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$
$\langle 4 \rangle 9.$ QED
BY $\langle 4 \rangle 2$, $\langle 4 \rangle 3$, $\langle 4 \rangle 4$, $\langle 4 \rangle 5$, $\langle 4 \rangle 6$, $\langle 4 \rangle 7$, $\langle 4 \rangle 8$ DEF $LL2TypeInvariant$
$\langle 3 \rangle 2.$ CASE $LL2PerformOperation$

For a $LL2PerformOperation$ action, we just walk through the definitions. Type safety follows directly.

$\langle 4 \rangle 1.$ PICK $input \in LL2AvailableInputs : LL2PerformOperation!(input)!1$
BY $\langle 3 \rangle 2$ DEF $LL2PerformOperation$
$\langle 4 \rangle$ $historySummaryHash \triangleq$
$Hash(LL2RAM.historySummary.anchor, LL2RAM.historySummary.extension)$
$\langle 4 \rangle$ $stateHash \triangleq Hash(LL2RAM.publicState, LL2RAM.privateStateEnc)$
$\langle 4 \rangle$ $historyStateBinding \triangleq Hash(historySummaryHash, stateHash)$
$\langle 4 \rangle$ $privateState \triangleq SymmetricDecrypt(LL2NVRAM.symmetricKey, LL2RAM.privateStateEnc)$
$\langle 4 \rangle$ $sResult \triangleq Service(LL2RAM.publicState, privateState, input)$
$\langle 4 \rangle$ $newPrivateStateEnc \triangleq$
$SymmetricEncrypt(LL2NVRAM.symmetricKey, sResult.newPrivateState)$
$\langle 4 \rangle$ $currentHistorySummary \triangleq [$
$anchor \mapsto LL2NVRAM.historySummaryAnchor,$
$extension \mapsto LL2SPCR]$
$\langle 4 \rangle$ $newHistorySummary \triangleq Successor(currentHistorySummary, input, LL2NVRAM.hashBarrier)$
$\langle 4 \rangle$ $newHistorySummaryHash \triangleq Hash(newHistorySummary.anchor, newHistorySummary.extension)$
$\langle 4 \rangle$ $newStateHash \triangleq Hash(sResult.newPublicState, newPrivateStateEnc)$
$\langle 4 \rangle$ $newHistoryStateBinding \triangleq Hash(newHistorySummaryHash, newStateHash)$
$\langle 4 \rangle$ $newAuthenticator \triangleq GenerateMAC(LL2NVRAM.symmetricKey, newHistoryStateBinding)$
$\langle 4 \rangle 2.$ $\land$ $historySummaryHash \in HashType$
$\land$ $stateHash \in HashType$
$\land$ $historyStateBinding \in HashType$
$\land$ $privateState \in PrivateStateType$
$\land$ $sResult \in ServiceResultType$
$\land$ $sResult.newPublicState \in PublicStateType$
$\land$ $sResult.newPrivateState \in PrivateStateType$
$\land$ $sResult.output \in OutputType$
$\land$ $newPrivateStateEnc \in PrivateStateEncType$
$\land$ $currentHistorySummary \in HistorySummaryType$
$\land$ $currentHistorySummary.anchor \in HashType$
$\land$ $currentHistorySummary.extension \in HashType$
$\land$ $newHistorySummary \in HistorySummaryType$
$\land$ $newHistorySummary.anchor \in HashType$
$\land$ $newHistorySummary.extension \in HashType$
$\land$ $newHistorySummaryHash \in HashType$
$\land$ $newStateHash \in HashType$
$\land$ $newHistoryStateBinding \in HashType$
$\land$ $newAuthenticator \in MACType$
$\langle 5 \rangle 1.$ $input \in LL2AvailableInputs$

BY $\langle 4\rangle 1$
$\langle 5\rangle 2$. $LL2TypeInvariant$
BY $\langle 2\rangle 1$
$\langle 5\rangle 3$. QED
BY $\langle 5\rangle 1$, $\langle 5\rangle 2$, $LL2PerformOperationDefsTypeSafeLemma$
$\langle 4\rangle$ HIDE DEF $historySummaryHash$, $stateHash$, $historyStateBinding$, $privateState$, $sResult$, $newPrivateStateEnc$, $currentHistorySummary$, $newHistorySummary$, $newHistorySummaryHash$, $newStateHash$, $newHistoryStateBinding$, $newAuthenticator$
$\langle 4\rangle 3$. $LL2AvailableInputs' \subseteq InputType$
$\langle 5\rangle 1$. $LL2AvailableInputs \subseteq InputType$
BY $\langle 2\rangle 1$ DEF $LL2TypeInvariant$
$\langle 5\rangle 2$. UNCHANGED $LL2AvailableInputs$
BY $\langle 4\rangle 1$
$\langle 5\rangle 3$. QED
BY $\langle 5\rangle 1$, $\langle 5\rangle 2$
$\langle 4\rangle 4$. $LL2ObservedOutputs' \subseteq OutputType$
$\langle 5\rangle 1$. $LL2ObservedOutputs \subseteq OutputType$
BY $\langle 2\rangle 1$ DEF $LL2TypeInvariant$
$\langle 5\rangle 2$. $LL2ObservedOutputs' = LL2ObservedOutputs \cup \{sResult.output\}$
BY $\langle 4\rangle 1$ DEF $sResult$, $privateState$
$\langle 5\rangle 3$. $sResult.output \in OutputType$
BY $\langle 4\rangle 2$
$\langle 5\rangle 4$. QED
BY $\langle 5\rangle 1$, $\langle 5\rangle 2$, $\langle 5\rangle 3$
$\langle 4\rangle 5$. $LL2ObservedAuthenticators' \subseteq MACType$
$\langle 5\rangle 1$. $LL2ObservedAuthenticators \subseteq MACType$
BY $\langle 2\rangle 1$ DEF $LL2TypeInvariant$
$\langle 5\rangle 2$. $LL2ObservedAuthenticators' = LL2ObservedAuthenticators \cup \{newAuthenticator\}$
BY $\langle 4\rangle 1$ DEF $newAuthenticator$, $newHistoryStateBinding$, $newHistorySummaryHash$, $newStateHash$, $newHistorySummary$, $currentHistorySummary$, $newPrivateStateEnc$, $sResult$, $privateState$
$\langle 5\rangle 3$. $newAuthenticator \in MACType$
BY $\langle 4\rangle 2$
$\langle 5\rangle 4$. QED
BY $\langle 5\rangle 1$, $\langle 5\rangle 2$, $\langle 5\rangle 3$
$\langle 4\rangle 6$. $LL2Disk' \in LL2UntrustedStorageType$
$\langle 5\rangle 1$. $LL2Disk \in LL2UntrustedStorageType$
BY $\langle 2\rangle 1$ DEF $LL2TypeInvariant$
$\langle 5\rangle 2$. UNCHANGED $LL2Disk$
BY $\langle 4\rangle 1$
$\langle 5\rangle 3$. QED
BY $\langle 5\rangle 1$, $\langle 5\rangle 2$
$\langle 4\rangle 7$. $LL2RAM' \in LL2UntrustedStorageType$
$\langle 5\rangle 1$. $LL2RAM' = [publicState \mapsto sResult.newPublicState, privateStateEnc \mapsto newPrivateStateEnc, historySummary \mapsto newHistorySummary, authenticator \mapsto newAuthenticator]$
BY $\langle 4\rangle 1$ DEF $newAuthenticator$, $newHistoryStateBinding$, $newHistorySummaryHash$, $newStateHash$, $newHistorySummary$, $currentHistorySummary$, $newPrivateStateEnc$, $sResult$, $privateState$
$\langle 5\rangle 2$. $sResult.newPublicState \in PublicStateType$

BY ⟨4⟩2

⟨5⟩3. $newPrivateStateEnc \in PrivateStateEncType$

BY ⟨4⟩2

⟨5⟩4. $newHistorySummary \in HistorySummaryType$

BY ⟨4⟩2 DEF $newHistorySummary$, $currentHistorySummary$

⟨5⟩5. $newAuthenticator \in MACType$

BY ⟨4⟩2

⟨5⟩6. QED

BY ⟨5⟩1, ⟨5⟩2, ⟨5⟩3, ⟨5⟩4, ⟨5⟩5 DEF $LL2UntrustedStorageType$

⟨4⟩8. $LL2NVRAM' \in LL2TrustedStorageType$

⟨5⟩1. $LL2NVRAM' = [$

$historySummaryAnchor \mapsto LL2NVRAM.historySummaryAnchor,$

$symmetricKey \mapsto LL2NVRAM.symmetricKey,$

$hashBarrier \mapsto LL2NVRAM.hashBarrier,$

$extensionInProgress \mapsto \text{TRUE}]$

BY ⟨4⟩1 DEF $LL2TypeInvariant$, $newHistorySummary$

⟨5⟩2. $LL2NVRAM \in LL2TrustedStorageType$

BY ⟨2⟩1 DEF $LL2TypeInvariant$

⟨5⟩3. TRUE ∈ BOOLEAN

OBVIOUS

⟨5⟩4. QED

BY ⟨5⟩1, ⟨5⟩2, ⟨5⟩3 DEF $LL2TrustedStorageType$

⟨4⟩9. $LL2SPCR' \in HashType$

⟨5⟩1. $LL2SPCR' = newHistorySummary.extension$

BY ⟨4⟩1 DEF $newHistorySummary$, $currentHistorySummary$

⟨5⟩2. $newHistorySummary.extension \in HashType$

BY ⟨4⟩2

⟨5⟩3. QED

BY ⟨5⟩1, ⟨5⟩2

⟨4⟩10. QED

BY ⟨4⟩3, ⟨4⟩4, ⟨4⟩5, ⟨4⟩6, ⟨4⟩7, ⟨4⟩8, ⟨4⟩9 DEF $LL2TypeInvariant$

⟨3⟩3. CASE $LL2RepeatOperation$

For a *LL2RepeatOperation* action, we just walk through the definitions. Type safety follows directly.

⟨4⟩1. PICK $input \in LL2AvailableInputs : LL2RepeatOperation!(input)!1$

BY ⟨3⟩3 DEF $LL2RepeatOperation$

⟨4⟩ $historySummaryHash \triangleq$

$Hash(LL2RAM.historySummary.anchor, LL2RAM.historySummary.extension)$

⟨4⟩ $stateHash \triangleq Hash(LL2RAM.publicState, LL2RAM.privateStateEnc)$

⟨4⟩ $historyStateBinding \triangleq Hash(historySummaryHash, stateHash)$

⟨4⟩ $newHistorySummary \triangleq Successor(LL2RAM.historySummary, input, LL2NVRAM.hashBarrier)$

⟨4⟩ $checkpointedHistorySummary \triangleq Checkpoint(LL2RAM.historySummary)$

⟨4⟩ $newCheckpointedHistorySummary \triangleq$

$Successor(checkpointedHistorySummary, input, LL2NVRAM.hashBarrier)$

⟨4⟩ $checkpointedNewHistorySummary \triangleq Checkpoint(newHistorySummary)$

⟨4⟩ $checkpointedNewCheckpointedHistorySummary \triangleq$

$Checkpoint(newCheckpointedHistorySummary)$

⟨4⟩ $privateState \triangleq SymmetricDecrypt(LL2NVRAM.symmetricKey, LL2RAM.privateStateEnc)$

⟨4⟩ $sResult \triangleq Service(LL2RAM.publicState, privateState, input)$

⟨4⟩ $newPrivateStateEnc \triangleq$

$SymmetricEncrypt(LL2NVRAM.symmetricKey, sResult.newPrivateState)$

⟨4⟩ $currentHistorySummary \triangleq [$

$anchor \mapsto LL2NVRAM.historySummaryAnchor,$

$extension \mapsto LL2SPCR]$
$\langle 4 \rangle$ $currentHistorySummaryHash \triangleq Hash(LL2NVRAM.historySummaryAnchor, LL2SPCR)$
$\langle 4 \rangle$ $newStateHash \triangleq Hash(sResult.newPublicState, newPrivateStateEnc)$
$\langle 4 \rangle$ $newHistoryStateBinding \triangleq Hash(currentHistorySummaryHash, newStateHash)$
$\langle 4 \rangle$ $newAuthenticator \triangleq GenerateMAC(LL2NVRAM.symmetricKey, newHistoryStateBinding)$
$\langle 4 \rangle 2.$ $\land\ historySummaryHash \in HashType$
$\land\ stateHash \in HashType$
$\land\ historyStateBinding \in HashType$
$\land\ newHistorySummary \in HistorySummaryType$
$\land\ newHistorySummary.anchor \in HashType$
$\land\ newHistorySummary.extension \in HashType$
$\land\ checkpointedHistorySummary \in HistorySummaryType$
$\land\ checkpointedHistorySummary.anchor \in HashType$
$\land\ checkpointedHistorySummary.extension \in HashType$
$\land\ newCheckpointedHistorySummary \in HistorySummaryType$
$\land\ newCheckpointedHistorySummary.anchor \in HashType$
$\land\ newCheckpointedHistorySummary.extension \in HashType$
$\land\ checkpointedNewHistorySummary \in HistorySummaryType$
$\land\ checkpointedNewHistorySummary.anchor \in HashType$
$\land\ checkpointedNewHistorySummary.extension \in HashType$
$\land\ checkpointedNewCheckpointedHistorySummary \in HistorySummaryType$
$\land\ checkpointedNewCheckpointedHistorySummary.anchor \in HashType$
$\land\ checkpointedNewCheckpointedHistorySummary.extension \in HashType$
$\land\ privateState \in PrivateStateType$
$\land\ sResult \in ServiceResultType$
$\land\ sResult.newPublicState \in PublicStateType$
$\land\ sResult.newPrivateState \in PrivateStateType$
$\land\ sResult.output \in OutputType$
$\land\ newPrivateStateEnc \in PrivateStateEncType$
$\land\ currentHistorySummary \in HistorySummaryType$
$\land\ currentHistorySummary.anchor \in HashType$
$\land\ currentHistorySummary.extension \in HashType$
$\land\ currentHistorySummaryHash \in HashType$
$\land\ newStateHash \in HashType$
$\land\ newHistoryStateBinding \in HashType$
$\land\ newAuthenticator \in MACType$
$\langle 5 \rangle 1.$ $input \in LL2AvailableInputs$
BY $\langle 4 \rangle 1$
$\langle 5 \rangle 2.$ $LL2TypeInvariant$
BY $\langle 2 \rangle 1$
$\langle 5 \rangle 3.$ QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$, $LL2RepeatOperationDefsTypeSafeLemma$
$\langle 4 \rangle$ HIDE DEF $historySummaryHash$, $stateHash$, $historyStateBinding$, $newHistorySummary$, $checkpointedHistorySummary$, $newCheckpointedHistorySummary$, $checkpointedNewHistorySummary$, $checkpointedNewCheckpointedHistorySummary$, $privateState$, $sResult$, $newPrivateStateEnc$, $currentHistorySummary$, $currentHistorySummaryHash$, $newStateHash$, $newHistoryStateBinding$, $newAuthenticator$
$\langle 4 \rangle 3.$ $LL2AvailableInputs' \subseteq InputType$
$\langle 5 \rangle 1.$ $LL2AvailableInputs \subseteq InputType$
BY $\langle 2 \rangle 1$ DEF $LL2TypeInvariant$
$\langle 5 \rangle 2.$ UNCHANGED $LL2AvailableInputs$

BY $\langle 4 \rangle 1$
$\langle 5 \rangle 3$. QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$
$\langle 4 \rangle 4$. $LL2ObservedOutputs' \subseteq OutputType$
$\langle 5 \rangle 1$. $LL2ObservedOutputs \subseteq OutputType$
BY $\langle 2 \rangle 1$ DEF $LL2TypeInvariant$
$\langle 5 \rangle 2$. $LL2ObservedOutputs' = LL2ObservedOutputs \cup \{sResult.output\}$
BY $\langle 4 \rangle 1$ DEF $sResult$, $privateState$
$\langle 5 \rangle 3$. $sResult.output \in OutputType$
BY $\langle 4 \rangle 2$
$\langle 5 \rangle 4$. QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$, $\langle 5 \rangle 3$
$\langle 4 \rangle 5$. $LL2ObservedAuthenticators' \subseteq MACType$
$\langle 5 \rangle 1$. $LL2ObservedAuthenticators \subseteq MACType$
BY $\langle 2 \rangle 1$ DEF $LL2TypeInvariant$
$\langle 5 \rangle 2$. $LL2ObservedAuthenticators' =$
$LL2ObservedAuthenticators \cup \{newAuthenticator\}$
BY $\langle 4 \rangle 1$ DEF $newAuthenticator$, $newHistoryStateBinding$, $currentHistorySummaryHash$, $newStateHash$, $newPrivateStateEnc$, $sResult$, $privateState$
$\langle 5 \rangle 3$. $newAuthenticator \in MACType$
BY $\langle 4 \rangle 2$
$\langle 5 \rangle 4$. QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$, $\langle 5 \rangle 3$
$\langle 4 \rangle 6$. $LL2Disk' \in LL2UntrustedStorageType$
$\langle 5 \rangle 1$. $LL2Disk \in LL2UntrustedStorageType$
BY $\langle 2 \rangle 1$ DEF $LL2TypeInvariant$
$\langle 5 \rangle 2$. UNCHANGED $LL2Disk$
BY $\langle 4 \rangle 1$
$\langle 5 \rangle 3$. QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$
$\langle 4 \rangle 7$. $LL2RAM' \in LL2UntrustedStorageType$
$\langle 5 \rangle 1$. $LL2RAM' = [publicState \mapsto sResult.newPublicState,$
$privateStateEnc \mapsto newPrivateStateEnc,$
$historySummary \mapsto currentHistorySummary,$
$authenticator \mapsto newAuthenticator]$
BY $\langle 4 \rangle 1$ DEF $newAuthenticator$, $newHistoryStateBinding$, $currentHistorySummaryHash$, $newStateHash$, $currentHistorySummary$, $newPrivateStateEnc$, $sResult$, $privateState$
$\langle 5 \rangle 2$. $sResult.newPublicState \in PublicStateType$
BY $\langle 4 \rangle 2$
$\langle 5 \rangle 3$. $newPrivateStateEnc \in PrivateStateEncType$
BY $\langle 4 \rangle 2$
$\langle 5 \rangle 4$. $currentHistorySummary \in HistorySummaryType$
BY $\langle 4 \rangle 2$
$\langle 5 \rangle 5$. $newAuthenticator \in MACType$
BY $\langle 4 \rangle 2$
$\langle 5 \rangle 6$. QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$, $\langle 5 \rangle 3$, $\langle 5 \rangle 4$, $\langle 5 \rangle 5$ DEF $LL2UntrustedStorageType$
$\langle 4 \rangle 8$. $LL2NVRAM' \in LL2TrustedStorageType$
$\langle 5 \rangle 1$. $LL2NVRAM \in LL2TrustedStorageType$
BY $\langle 2 \rangle 1$ DEF $LL2TypeInvariant$
$\langle 5 \rangle 2$. UNCHANGED $LL2NVRAM$

BY ⟨4⟩1

⟨5⟩3. QED

BY ⟨5⟩1, ⟨5⟩2

⟨4⟩9. $LL2SPCR' \in HashType$

⟨5⟩1. $LL2SPCR \in HashType$

BY ⟨2⟩1 DEF $LL2TypeInvariant$

⟨5⟩2. UNCHANGED $LL2SPCR$

BY ⟨4⟩1

⟨5⟩3. QED

BY ⟨5⟩1, ⟨5⟩2

⟨4⟩10. QED

BY ⟨4⟩3, ⟨4⟩4, ⟨4⟩5, ⟨4⟩6, ⟨4⟩7, ⟨4⟩8, ⟨4⟩9 DEF $LL2TypeInvariant$

⟨3⟩4. CASE $LL2TakeCheckpoint$

For a *LL2TakeCheckpoint* action, we just walk through the definitions. Type safety follows directly.

⟨4⟩ $newHistorySummaryAnchor \triangleq Hash(LL2NVRAM.historySummaryAnchor, LL2SPCR)$

⟨4⟩1. $newHistorySummaryAnchor \in HashType$

⟨5⟩1. $LL2TypeInvariant$

BY ⟨2⟩1

⟨5⟩2. QED

BY ⟨5⟩1, $LL2TakeCheckpointDefsTypeSafeLemma$

⟨4⟩ HIDE DEF $newHistorySummaryAnchor$

⟨4⟩2. $LL2AvailableInputs' \subseteq InputType$

⟨5⟩1. $LL2AvailableInputs \subseteq InputType$

BY ⟨2⟩1 DEF $LL2TypeInvariant$

⟨5⟩2. UNCHANGED $LL2AvailableInputs$

BY ⟨3⟩4 DEF $LL2TakeCheckpoint$

⟨5⟩3. QED

BY ⟨5⟩1, ⟨5⟩2

⟨4⟩3. $LL2ObservedOutputs' \subseteq OutputType$

⟨5⟩1. $LL2ObservedOutputs \subseteq OutputType$

BY ⟨2⟩1 DEF $LL2TypeInvariant$

⟨5⟩2. UNCHANGED $LL2ObservedOutputs$

BY ⟨3⟩4 DEF $LL2TakeCheckpoint$

⟨5⟩3. QED

BY ⟨5⟩1, ⟨5⟩2

⟨4⟩4. $LL2ObservedAuthenticators' \subseteq MACType$

⟨5⟩1. $LL2ObservedAuthenticators \subseteq MACType$

BY ⟨2⟩1 DEF $LL2TypeInvariant$

⟨5⟩2. UNCHANGED $LL2ObservedAuthenticators$

BY ⟨3⟩4 DEF $LL2TakeCheckpoint$

⟨5⟩3. QED

BY ⟨5⟩1, ⟨5⟩2

⟨4⟩5. $LL2Disk' \in LL2UntrustedStorageType$

⟨5⟩1. $LL2Disk \in LL2UntrustedStorageType$

BY ⟨2⟩1 DEF $LL2TypeInvariant$

⟨5⟩2. UNCHANGED $LL2Disk$

BY ⟨3⟩4 DEF $LL2TakeCheckpoint$

⟨5⟩3. QED

BY ⟨5⟩1, ⟨5⟩2

⟨4⟩6. $LL2RAM' \in LL2UntrustedStorageType$

⟨5⟩1. $LL2RAM \in LL2UntrustedStorageType$

BY ⟨2⟩1 DEF $LL2TypeInvariant$

$\langle 5\rangle 2.$ UNCHANGED $\mathit{LL2RAM}$
BY $\langle 3\rangle 4$ DEF $\mathit{LL2TakeCheckpoint}$
$\langle 5\rangle 3.$ QED
BY $\langle 5\rangle 1$, $\langle 5\rangle 2$
$\langle 4\rangle 7.$ $\mathit{LL2NVRAM}' \in \mathit{LL2TrustedStorageType}$
$\langle 5\rangle 1.$ $\mathit{LL2NVRAM} \in \mathit{LL2TrustedStorageType}$
BY $\langle 2\rangle 1$ DEF $\mathit{LL2TypeInvariant}$
$\langle 5\rangle 2.$ $\mathit{LL2NVRAM}' = [$
$\mathit{historySummaryAnchor} \mapsto \mathit{newHistorySummaryAnchor}$,
$\mathit{symmetricKey} \mapsto \mathit{LL2NVRAM.symmetricKey}$,
$\mathit{hashBarrier} \mapsto \mathit{LL2NVRAM.hashBarrier}$,
$\mathit{extensionInProgress} \mapsto \text{FALSE}]$
BY $\langle 3\rangle 4$ DEF $\mathit{LL2TakeCheckpoint}$, $\mathit{newHistorySummaryAnchor}$
$\langle 5\rangle 3.$ $\mathit{newHistorySummaryAnchor} \in \mathit{HashType}$
BY $\langle 4\rangle 1$
$\langle 5\rangle 4.$ FALSE $\in$ BOOLEAN
OBVIOUS
$\langle 5\rangle 5.$ QED
BY $\langle 5\rangle 1$, $\langle 5\rangle 2$, $\langle 5\rangle 3$, $\langle 5\rangle 4$ DEF $\mathit{LL2TrustedStorageType}$
$\langle 4\rangle 8.$ $\mathit{LL2SPCR}' \in \mathit{HashType}$
$\langle 5\rangle 1.$ $\mathit{LL2SPCR} \in \mathit{HashType}$
BY $\langle 2\rangle 1$ DEF $\mathit{LL2TypeInvariant}$
$\langle 5\rangle 2.$ UNCHANGED $\mathit{LL2SPCR}$
BY $\langle 3\rangle 4$ DEF $\mathit{LL2TakeCheckpoint}$
$\langle 5\rangle 3.$ QED
BY $\langle 5\rangle 1$, $\langle 5\rangle 2$
$\langle 4\rangle 9.$ QED
BY $\langle 4\rangle 2$, $\langle 4\rangle 3$, $\langle 4\rangle 4$, $\langle 4\rangle 5$, $\langle 4\rangle 6$, $\langle 4\rangle 7$, $\langle 4\rangle 8$ DEF $\mathit{LL2TypeInvariant}$
$\langle 3\rangle 5.$ CASE $\mathit{LL2Restart}$

For a *LL2Restart* action, we just walk through the definitions. Type safety follows directly.

$\langle 4\rangle 1.$ PICK $\mathit{untrustedStorage} \in \mathit{LL2UntrustedStorageType}$,
$\mathit{randomSymmetricKey} \in \mathit{SymmetricKeyType} \setminus \{\mathit{LL2NVRAM.symmetricKey}\}$,
$\mathit{hash} \in \mathit{HashType}$:
$\mathit{LL2Restart}!(\mathit{untrustedStorage}, \mathit{randomSymmetricKey}, \mathit{hash})$
BY $\langle 3\rangle 5$ DEF $\mathit{LL2Restart}$
$\langle 4\rangle 2.$ $\mathit{LL2AvailableInputs}' \subseteq \mathit{InputType}$
$\langle 5\rangle 1.$ $\mathit{LL2AvailableInputs} \subseteq \mathit{InputType}$
BY $\langle 2\rangle 1$ DEF $\mathit{LL2TypeInvariant}$
$\langle 5\rangle 2.$ UNCHANGED $\mathit{LL2AvailableInputs}$
BY $\langle 4\rangle 1$
$\langle 5\rangle 3.$ QED
BY $\langle 5\rangle 1$, $\langle 5\rangle 2$
$\langle 4\rangle 3.$ $\mathit{LL2ObservedOutputs}' \subseteq \mathit{OutputType}$
$\langle 5\rangle 1.$ $\mathit{LL2ObservedOutputs} \subseteq \mathit{OutputType}$
BY $\langle 2\rangle 1$ DEF $\mathit{LL2TypeInvariant}$
$\langle 5\rangle 2.$ UNCHANGED $\mathit{LL2ObservedOutputs}$
BY $\langle 4\rangle 1$
$\langle 5\rangle 3.$ QED
BY $\langle 5\rangle 1$, $\langle 5\rangle 2$
$\langle 4\rangle 4.$ $\mathit{LL2ObservedAuthenticators}' \subseteq \mathit{MACType}$
$\langle 5\rangle 1.$ $\mathit{LL2ObservedAuthenticators} \subseteq \mathit{MACType}$
BY $\langle 2\rangle 1$ DEF $\mathit{LL2TypeInvariant}$

$\langle 5\rangle 2.$ UNCHANGED $LL2ObservedAuthenticators$
BY $\langle 4\rangle 1$
$\langle 5\rangle 3.$ QED
BY $\langle 5\rangle 1$, $\langle 5\rangle 2$
$\langle 4\rangle 5.$ $LL2Disk' \in LL2UntrustedStorageType$
$\langle 5\rangle 1.$ $LL2Disk \in LL2UntrustedStorageType$
BY $\langle 2\rangle 1$ DEF $LL2TypeInvariant$
$\langle 5\rangle 2.$ UNCHANGED $LL2Disk$
BY $\langle 4\rangle 1$
$\langle 5\rangle 3.$ QED
BY $\langle 5\rangle 1$, $\langle 5\rangle 2$
$\langle 4\rangle 6.$ $LL2RAM' \in LL2UntrustedStorageType$
$\langle 5\rangle 1.$ $LL2Disk \in LL2UntrustedStorageType$
BY $\langle 2\rangle 1$ DEF $LL2TypeInvariant$
$\langle 5\rangle 2.$ $LL2RAM' = untrustedStorage$
BY $\langle 4\rangle 1$
$\langle 5\rangle 3.$ $untrustedStorage \in LL2UntrustedStorageType$
BY $\langle 4\rangle 1$
$\langle 5\rangle 4.$ QED
BY $\langle 5\rangle 1$, $\langle 5\rangle 2$, $\langle 5\rangle 3$
$\langle 4\rangle 7.$ $LL2NVRAM' \in LL2TrustedStorageType$
$\langle 5\rangle 1.$ $LL2NVRAM \in LL2TrustedStorageType$
BY $\langle 2\rangle 1$ DEF $LL2TypeInvariant$
$\langle 5\rangle 2.$ UNCHANGED $LL2NVRAM$
BY $\langle 4\rangle 1$
$\langle 5\rangle 3.$ QED
BY $\langle 5\rangle 1$, $\langle 5\rangle 2$
$\langle 4\rangle 8.$ $LL2SPCR' \in HashType$
$\langle 5\rangle 1.$ $LL2SPCR' = BaseHashValue$
BY $\langle 4\rangle 1$
$\langle 5\rangle 2.$ $BaseHashValue \in HashType$
BY $BaseHashValueTypeSafe$
$\langle 5\rangle 3.$ QED
BY $\langle 5\rangle 1$, $\langle 5\rangle 2$
$\langle 4\rangle 9.$ QED
BY $\langle 4\rangle 2$, $\langle 4\rangle 3$, $\langle 4\rangle 4$, $\langle 4\rangle 5$, $\langle 4\rangle 6$, $\langle 4\rangle 7$, $\langle 4\rangle 8$ DEF $LL2TypeInvariant$
$\langle 3\rangle 6.$ CASE $LL2ReadDisk$

Type safety is straightforward for a *LL2ReadDisk* action.

$\langle 4\rangle 1.$ $LL2AvailableInputs' \subseteq InputType$
$\langle 5\rangle 1.$ $LL2AvailableInputs \subseteq InputType$
BY $\langle 2\rangle 1$ DEF $LL2TypeInvariant$
$\langle 5\rangle 2.$ UNCHANGED $LL2AvailableInputs$
BY $\langle 3\rangle 6$ DEF $LL2ReadDisk$
$\langle 5\rangle 3.$ QED
BY $\langle 5\rangle 1$, $\langle 5\rangle 2$
$\langle 4\rangle 2.$ $LL2ObservedOutputs' \subseteq OutputType$
$\langle 5\rangle 1.$ $LL2ObservedOutputs \subseteq OutputType$
BY $\langle 2\rangle 1$ DEF $LL2TypeInvariant$
$\langle 5\rangle 2.$ UNCHANGED $LL2ObservedOutputs$
BY $\langle 3\rangle 6$ DEF $LL2ReadDisk$
$\langle 5\rangle 3.$ QED
BY $\langle 5\rangle 1$, $\langle 5\rangle 2$

⟨4⟩3. $LL2ObservedAuthenticators' \subseteq MACType$
⟨5⟩1. $LL2ObservedAuthenticators \subseteq MACType$
BY ⟨2⟩1 DEF $LL2TypeInvariant$
⟨5⟩2. UNCHANGED $LL2ObservedAuthenticators$
BY ⟨3⟩6 DEF $LL2ReadDisk$
⟨5⟩3. QED
BY ⟨5⟩1, ⟨5⟩2
⟨4⟩4. $LL2Disk' \in LL2UntrustedStorageType$
⟨5⟩1. $LL2Disk \in LL2UntrustedStorageType$
BY ⟨2⟩1 DEF $LL2TypeInvariant$
⟨5⟩2. UNCHANGED $LL2Disk$
BY ⟨3⟩6 DEF $LL2ReadDisk$
⟨5⟩3. QED
BY ⟨5⟩1, ⟨5⟩2
⟨4⟩5. $LL2RAM' \in LL2UntrustedStorageType$
⟨5⟩1. $LL2Disk \in LL2UntrustedStorageType$
BY ⟨2⟩1 DEF $LL2TypeInvariant$
⟨5⟩2. $LL2RAM' = LL2Disk$
BY ⟨3⟩6 DEF $LL2ReadDisk$
⟨5⟩3. QED
BY ⟨5⟩1, ⟨5⟩2
⟨4⟩6. $LL2NVRAM' \in LL2TrustedStorageType$
⟨5⟩1. $LL2NVRAM \in LL2TrustedStorageType$
BY ⟨2⟩1 DEF $LL2TypeInvariant$
⟨5⟩2. UNCHANGED $LL2NVRAM$
BY ⟨3⟩6 DEF $LL2ReadDisk$
⟨5⟩3. QED
BY ⟨5⟩1, ⟨5⟩2
⟨4⟩7. $LL2SPCR' \in HashType$
⟨5⟩1. $LL2SPCR \in HashType$
BY ⟨2⟩1 DEF $LL2TypeInvariant$
⟨5⟩2. UNCHANGED $LL2SPCR$
BY ⟨3⟩6 DEF $LL2ReadDisk$
⟨5⟩3. QED
BY ⟨5⟩1, ⟨5⟩2
⟨4⟩8. QED
BY ⟨4⟩1, ⟨4⟩2, ⟨4⟩3, ⟨4⟩4, ⟨4⟩5, ⟨4⟩6, ⟨4⟩7 DEF $LL2TypeInvariant$
⟨3⟩7. CASE $LL2WriteDisk$

Type safety is straightforward for a *LL2WriteDisk* action.

⟨4⟩1. $LL2AvailableInputs' \subseteq InputType$
⟨5⟩1. $LL2AvailableInputs \subseteq InputType$
BY ⟨2⟩1 DEF $LL2TypeInvariant$
⟨5⟩2. UNCHANGED $LL2AvailableInputs$
BY ⟨3⟩7 DEF $LL2WriteDisk$
⟨5⟩3. QED
BY ⟨5⟩1, ⟨5⟩2
⟨4⟩2. $LL2ObservedOutputs' \subseteq OutputType$
⟨5⟩1. $LL2ObservedOutputs \subseteq OutputType$
BY ⟨2⟩1 DEF $LL2TypeInvariant$
⟨5⟩2. UNCHANGED $LL2ObservedOutputs$
BY ⟨3⟩7 DEF $LL2WriteDisk$
⟨5⟩3. QED

BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$
$\langle 4 \rangle 3$. $LL2ObservedAuthenticators' \subseteq MACType$
$\langle 5 \rangle 1$. $LL2ObservedAuthenticators \subseteq MACType$
BY $\langle 2 \rangle 1$ DEF $LL2TypeInvariant$
$\langle 5 \rangle 2$. UNCHANGED $LL2ObservedAuthenticators$
BY $\langle 3 \rangle 7$ DEF $LL2WriteDisk$
$\langle 5 \rangle 3$. QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$
$\langle 4 \rangle 4$. $LL2Disk' \in LL2UntrustedStorageType$
$\langle 5 \rangle 1$. $LL2RAM \in LL2UntrustedStorageType$
BY $\langle 2 \rangle 1$ DEF $LL2TypeInvariant$
$\langle 5 \rangle 2$. $LL2Disk' = LL2RAM$
BY $\langle 3 \rangle 7$ DEF $LL2WriteDisk$
$\langle 5 \rangle 3$. QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$
$\langle 4 \rangle 5$. $LL2RAM' \in LL2UntrustedStorageType$
$\langle 5 \rangle 1$. $LL2RAM \in LL2UntrustedStorageType$
BY $\langle 2 \rangle 1$ DEF $LL2TypeInvariant$
$\langle 5 \rangle 2$. UNCHANGED $LL2RAM$
BY $\langle 3 \rangle 7$ DEF $LL2WriteDisk$
$\langle 5 \rangle 3$. QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$
$\langle 4 \rangle 6$. $LL2NVRAM' \in LL2TrustedStorageType$
$\langle 5 \rangle 1$. $LL2NVRAM \in LL2TrustedStorageType$
BY $\langle 2 \rangle 1$ DEF $LL2TypeInvariant$
$\langle 5 \rangle 2$. UNCHANGED $LL2NVRAM$
BY $\langle 3 \rangle 7$ DEF $LL2WriteDisk$
$\langle 5 \rangle 3$. QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$
$\langle 4 \rangle 7$. $LL2SPCR' \in HashType$
$\langle 5 \rangle 1$. $LL2SPCR \in HashType$
BY $\langle 2 \rangle 1$ DEF $LL2TypeInvariant$
$\langle 5 \rangle 2$. UNCHANGED $LL2SPCR$
BY $\langle 3 \rangle 7$ DEF $LL2WriteDisk$
$\langle 5 \rangle 3$. QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$
$\langle 4 \rangle 8$. QED
BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$, $\langle 4 \rangle 3$, $\langle 4 \rangle 4$, $\langle 4 \rangle 5$, $\langle 4 \rangle 6$, $\langle 4 \rangle 7$ DEF $LL2TypeInvariant$
$\langle 3 \rangle 8$. CASE $LL2CorruptRAM$

Type safety is straightforward for a $LL2CorruptRAM$ action.

$\langle 4 \rangle 1$. PICK $untrustedStorage \in LL2UntrustedStorageType$,
$fakeSymmetricKey \in SymmetricKeyType \setminus \{LL2NVRAM.symmetricKey\}$,
$hash \in HashType$:
$LL2CorruptRAM!(untrustedStorage, fakeSymmetricKey, hash)$
BY $\langle 3 \rangle 8$ DEF $LL2CorruptRAM$
$\langle 4 \rangle 2$. $LL2AvailableInputs' \subseteq InputType$
$\langle 5 \rangle 1$. $LL2AvailableInputs \subseteq InputType$
BY $\langle 2 \rangle 1$ DEF $LL2TypeInvariant$
$\langle 5 \rangle 2$. UNCHANGED $LL2AvailableInputs$
BY $\langle 4 \rangle 1$
$\langle 5 \rangle 3$. QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$

$\langle 4 \rangle 3.$ $LL2ObservedOutputs' \subseteq OutputType$
$\langle 5 \rangle 1.$ $LL2ObservedOutputs \subseteq OutputType$
BY $\langle 2 \rangle 1$ DEF $LL2TypeInvariant$
$\langle 5 \rangle 2.$ UNCHANGED $LL2ObservedOutputs$
BY $\langle 4 \rangle 1$
$\langle 5 \rangle 3.$ QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$
$\langle 4 \rangle 4.$ $LL2ObservedAuthenticators' \subseteq MACType$
$\langle 5 \rangle 1.$ $LL2ObservedAuthenticators \subseteq MACType$
BY $\langle 2 \rangle 1$ DEF $LL2TypeInvariant$
$\langle 5 \rangle 2.$ UNCHANGED $LL2ObservedAuthenticators$
BY $\langle 4 \rangle 1$
$\langle 5 \rangle 3.$ QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$
$\langle 4 \rangle 5.$ $LL2Disk' \in LL2UntrustedStorageType$
$\langle 5 \rangle 1.$ $LL2Disk \in LL2UntrustedStorageType$
BY $\langle 2 \rangle 1$ DEF $LL2TypeInvariant$
$\langle 5 \rangle 2.$ UNCHANGED $LL2Disk$
BY $\langle 4 \rangle 1$
$\langle 5 \rangle 3.$ QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$
$\langle 4 \rangle 6.$ $LL2RAM' \in LL2UntrustedStorageType$
$\langle 5 \rangle 1.$ $untrustedStorage \in LL2UntrustedStorageType$
BY $\langle 4 \rangle 1$
$\langle 5 \rangle 2.$ $LL2RAM' = untrustedStorage$
BY $\langle 4 \rangle 1$
$\langle 5 \rangle 3.$ QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$
$\langle 4 \rangle 7.$ $LL2NVRAM' \in LL2TrustedStorageType$
$\langle 5 \rangle 1.$ $LL2NVRAM \in LL2TrustedStorageType$
BY $\langle 2 \rangle 1$ DEF $LL2TypeInvariant$
$\langle 5 \rangle 2.$ UNCHANGED $LL2NVRAM$
BY $\langle 4 \rangle 1$
$\langle 5 \rangle 3.$ QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$
$\langle 4 \rangle 8.$ $LL2SPCR' \in HashType$
$\langle 5 \rangle 1.$ $LL2SPCR \in HashType$
BY $\langle 2 \rangle 1$ DEF $LL2TypeInvariant$
$\langle 5 \rangle 2.$ UNCHANGED $LL2SPCR$
BY $\langle 4 \rangle 1$
$\langle 5 \rangle 3.$ QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$
$\langle 4 \rangle 9.$ QED
BY $\langle 4 \rangle 2$, $\langle 4 \rangle 3$, $\langle 4 \rangle 4$, $\langle 4 \rangle 5$, $\langle 4 \rangle 6$, $\langle 4 \rangle 7$, $\langle 4 \rangle 8$ DEF $LL2TypeInvariant$
$\langle 3 \rangle 9.$ CASE $LL2CorruptSPCR$

For a $LL2CorruptSPCR$ action, we just walk through the definitions. Type safety follows directly.

$\langle 4 \rangle 1.$ PICK $fakeHash \in HashDomain : LL2CorruptSPCR!(fakeHash)!1$
BY $\langle 3 \rangle 9$ DEF $LL2CorruptSPCR$
$\langle 4 \rangle$ $newHistorySummaryExtension \triangleq Hash(LL2SPCR, fakeHash)$
$\langle 4 \rangle 2.$ $newHistorySummaryExtension \in HashType$
$\langle 5 \rangle 1.$ $LL2TypeInvariant$
BY $\langle 2 \rangle 1$

$\langle 5\rangle 2.$ QED

BY $\langle 5\rangle 1$, *LL2CorruptSPCRDefsTypeSafeLemma*

$\langle 4\rangle$ HIDE DEF *newHistorySummaryExtension*

$\langle 4\rangle 3.$ $\mathit{LL2AvailableInputs}' \subseteq \mathit{InputType}$

$\langle 5\rangle 1.$ $\mathit{LL2AvailableInputs} \subseteq \mathit{InputType}$

BY $\langle 2\rangle 1$ DEF *LL2TypeInvariant*

$\langle 5\rangle 2.$ UNCHANGED *LL2AvailableInputs*

BY $\langle 4\rangle 1$

$\langle 5\rangle 3.$ QED

BY $\langle 5\rangle 1$, $\langle 5\rangle 2$

$\langle 4\rangle 4.$ $\mathit{LL2ObservedOutputs}' \subseteq \mathit{OutputType}$

$\langle 5\rangle 1.$ $\mathit{LL2ObservedOutputs} \subseteq \mathit{OutputType}$

BY $\langle 2\rangle 1$ DEF *LL2TypeInvariant*

$\langle 5\rangle 2.$ UNCHANGED *LL2ObservedOutputs*

BY $\langle 4\rangle 1$

$\langle 5\rangle 3.$ QED

BY $\langle 5\rangle 1$, $\langle 5\rangle 2$

$\langle 4\rangle 5.$ $\mathit{LL2ObservedAuthenticators}' \subseteq \mathit{MACType}$

$\langle 5\rangle 1.$ $\mathit{LL2ObservedAuthenticators} \subseteq \mathit{MACType}$

BY $\langle 2\rangle 1$ DEF *LL2TypeInvariant*

$\langle 5\rangle 2.$ UNCHANGED *LL2ObservedAuthenticators*

BY $\langle 4\rangle 1$

$\langle 5\rangle 3.$ QED

BY $\langle 5\rangle 1$, $\langle 5\rangle 2$

$\langle 4\rangle 6.$ $\mathit{LL2Disk}' \in \mathit{LL2UntrustedStorageType}$

$\langle 5\rangle 1.$ $\mathit{LL2Disk} \in \mathit{LL2UntrustedStorageType}$

BY $\langle 2\rangle 1$ DEF *LL2TypeInvariant*

$\langle 5\rangle 2.$ UNCHANGED *LL2Disk*

BY $\langle 4\rangle 1$

$\langle 5\rangle 3.$ QED

BY $\langle 5\rangle 1$, $\langle 5\rangle 2$

$\langle 4\rangle 7.$ $\mathit{LL2RAM}' \in \mathit{LL2UntrustedStorageType}$

$\langle 5\rangle 1.$ $\mathit{LL2RAM} \in \mathit{LL2UntrustedStorageType}$

BY $\langle 2\rangle 1$ DEF *LL2TypeInvariant*

$\langle 5\rangle 2.$ UNCHANGED *LL2RAM*

BY $\langle 4\rangle 1$

$\langle 5\rangle 3.$ QED

BY $\langle 5\rangle 1$, $\langle 5\rangle 2$

$\langle 4\rangle 8.$ $\mathit{LL2NVRAM}' \in \mathit{LL2TrustedStorageType}$

$\langle 5\rangle 1.$ $\mathit{LL2NVRAM} \in \mathit{LL2TrustedStorageType}$

BY $\langle 2\rangle 1$ DEF *LL2TypeInvariant*

$\langle 5\rangle 2.$ UNCHANGED *LL2NVRAM*

BY $\langle 4\rangle 1$

$\langle 5\rangle 3.$ QED

BY $\langle 5\rangle 1$, $\langle 5\rangle 2$

$\langle 4\rangle 9.$ $\mathit{LL2SPCR}' \in \mathit{HashType}$

$\langle 5\rangle 1.$ $\mathit{newHistorySummaryExtension} \in \mathit{HashType}$

BY $\langle 4\rangle 2$

$\langle 5\rangle 2.$ $\mathit{LL2SPCR}' = \mathit{newHistorySummaryExtension}$

BY $\langle 4\rangle 1$ DEF *newHistorySummaryExtension*

$\langle 5\rangle 3.$ QED

BY $\langle 5\rangle 1$, $\langle 5\rangle 2$

$\langle 4 \rangle 10$. QED

BY $\langle 4 \rangle 3$, $\langle 4 \rangle 4$, $\langle 4 \rangle 5$, $\langle 4 \rangle 6$, $\langle 4 \rangle 7$, $\langle 4 \rangle 8$, $\langle 4 \rangle 9$ DEF *LL2TypeInvariant*

$\langle 3 \rangle 10$. QED

BY $\langle 2 \rangle 3$, $\langle 3 \rangle 1$, $\langle 3 \rangle 2$, $\langle 3 \rangle 3$, $\langle 3 \rangle 4$, $\langle 3 \rangle 5$, $\langle 3 \rangle 6$, $\langle 3 \rangle 7$, $\langle 3 \rangle 8$, $\langle 3 \rangle 9$ DEF *LL2Next*

$\langle 2 \rangle 4$. QED

BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, $\langle 2 \rangle 3$

$\langle 1 \rangle 3$. QED

Using the *Inv*1 proof rule, the base case and the induction step together imply that the invariant always holds.

$\langle 2 \rangle 1$. $LL2TypeInvariant \land \Box[LL2Next]_{LL2Vars} \Rightarrow \Box LL2TypeInvariant$

BY $\langle 1 \rangle 2$, *Inv*1

$\langle 2 \rangle 2$. QED

BY $\langle 2 \rangle 1$, $\langle 1 \rangle 1$ DEF *LL2Spec*

4.10 Proofs of Lemmas Relating to the Memoir-Opt Refinement

MODULE *MemoirLL2RefinementLemmas*

This module states and proves several lemmas about the operators defined in the *LL2Refinement* module.

This module includes the following theorems:
HistorySummaryRecordCompositionLemma
LL1DiskRecordCompositionLemma
LL1RAMRecordCompositionLemma
LL1NVRAMRecordCompositionLemma
CheckpointHasBaseExtensionLemma
SuccessorHasNonBaseExtensionLemma
HistorySummariesMatchUniqueLemma
AuthenticatorsMatchUniqueLemma
AuthenticatorSetsMatchUniqueLemma
LL2NVRAMLogicalHistorySummaryTypeSafe
AuthenticatorInSetLemma
AuthenticatorGeneratedLemma
AuthenticatorValidatedLemma
HistorySummariesMatchAcrossCheckpointLemma

EXTENDS *MemoirLL2TypeSafety*

The *HistorySummaryRecordCompositionLemma* is a formality needed for the prover to compose a *HistorySummary* record from fields of the appropriate type.

THEOREM $HistorySummaryRecordCompositionLemma \triangleq$
$\quad \forall historySummary \in HistorySummaryType :$
$\quad\quad historySummary = [$
$\quad\quad\quad anchor \mapsto historySummary.anchor,$
$\quad\quad\quad extension \mapsto historySummary.extension]$
$\langle 1 \rangle 1.$ TAKE $historySummary \in HistorySummaryType$
$\langle 1 \rangle\ historysummary \triangleq [$
$\quad\quad anchor \mapsto historySummary.anchor,$
$\quad\quad extension \mapsto historySummary.extension]$
$\langle 1 \rangle 2.\ historysummary = [i \in \{\text{“anchor”}, \text{“extension”}\} \mapsto historysummary[i]]$
OBVIOUS
$\langle 1 \rangle 3.\ historySummary = [i \in \{\text{“anchor”}, \text{“extension”}\} \mapsto historySummary[i]]$
BY $\langle 1 \rangle 1$ DEF $HistorySummaryType$
$\langle 1 \rangle 4.$ QED
BY $\langle 1 \rangle 2$, $\langle 1 \rangle 3$

The *LL1DiskRecordCompositionLemma* is a formality needed for the prover to compose an *LL1Disk* record from fields of the appropriate type.

THEOREM $LL1DiskRecordCompositionLemma \triangleq$
$\quad \land \quad LL1Disk \in LL1UntrustedStorageType \Rightarrow$
$\quad\quad\quad LL1Disk = [$
$\quad\quad\quad\quad publicState \mapsto LL1Disk.publicState,$
$\quad\quad\quad\quad privateStateEnc \mapsto LL1Disk.privateStateEnc,$
$\quad\quad\quad\quad historySummary \mapsto LL1Disk.historySummary,$
$\quad\quad\quad\quad authenticator \mapsto LL1Disk.authenticator]$
$\quad \land \quad LL1Disk' \in LL1UntrustedStorageType \Rightarrow$
$\quad\quad\quad LL1Disk' = [$

$publicState \mapsto LL1Disk.publicState',$
$privateStateEnc \mapsto LL1Disk.privateStateEnc',$
$historySummary \mapsto LL1Disk.historySummary',$
$authenticator \mapsto LL1Disk.authenticator']$

$\langle 1 \rangle 1.$ $LL1Disk \in LL1UntrustedStorageType \Rightarrow$
$LL1Disk = [$
$publicState \mapsto LL1Disk.publicState,$
$privateStateEnc \mapsto LL1Disk.privateStateEnc,$
$historySummary \mapsto LL1Disk.historySummary,$
$authenticator \mapsto LL1Disk.authenticator]$

$\langle 2 \rangle 1.$ HAVE $LL1Disk \in LL1UntrustedStorageType$

$\langle 2 \rangle$ $ll1disk \triangleq [$
$publicState \mapsto LL1Disk.publicState,$
$privateStateEnc \mapsto LL1Disk.privateStateEnc,$
$historySummary \mapsto LL1Disk.historySummary,$
$authenticator \mapsto LL1Disk.authenticator]$

$\langle 2 \rangle 2.$ $ll1disk =$
$[i \in \{$ "publicState", "privateStateEnc", "historySummary", "authenticator" $\} \mapsto ll1disk[i]]$

OBVIOUS

$\langle 2 \rangle 3.$ $LL1Disk =$
$[i \in \{$ "publicState", "privateStateEnc", "historySummary", "authenticator" $\} \mapsto LL1Disk[i]]$

BY $\langle 2 \rangle 1$ DEF $LL1UntrustedStorageType$

$\langle 2 \rangle 4.$ QED

BY $\langle 2 \rangle 2$, $\langle 2 \rangle 3$

$\langle 1 \rangle 2.$ $LL1Disk' \in LL1UntrustedStorageType \Rightarrow$
$LL1Disk' = [$
$publicState \mapsto LL1Disk.publicState',$
$privateStateEnc \mapsto LL1Disk.privateStateEnc',$
$historySummary \mapsto LL1Disk.historySummary',$
$authenticator \mapsto LL1Disk.authenticator']$

$\langle 2 \rangle 1.$ HAVE $LL1Disk' \in LL1UntrustedStorageType$

$\langle 2 \rangle$ $ll1diskprime \triangleq [$
$publicState \mapsto LL1Disk.publicState',$
$privateStateEnc \mapsto LL1Disk.privateStateEnc',$
$historySummary \mapsto LL1Disk.historySummary',$
$authenticator \mapsto LL1Disk.authenticator']$

$\langle 2 \rangle 2.$ $ll1diskprime =$
$[i \in \{$ "publicState", "privateStateEnc", "historySummary", "authenticator" $\} \mapsto ll1diskprime[i]]$

OBVIOUS

$\langle 2 \rangle 3.$ $LL1Disk' =$

$[i \in \{\text{"publicState"},$
$\quad \text{"privateStateEnc"},$
$\quad \text{"historySummary"},$
$\quad \text{"authenticator"}\}$
$\quad \mapsto LL1Disk[i]']$

BY ⟨2⟩1 DEF $LL1UntrustedStorageType$

⟨2⟩4. QED

BY ⟨2⟩2, ⟨2⟩3

⟨1⟩3. QED

BY ⟨1⟩1, ⟨1⟩2

The $LL1RAMRecordCompositionLemma$ is a formality needed for the prover to compose an $LL1RAM$ record from fields of the appropriate type.

THEOREM $LL1RAMRecordCompositionLemma \triangleq$

$\wedge \quad LL1RAM \in LL1UntrustedStorageType \Rightarrow$
$\quad LL1RAM = [$
$\quad\quad publicState \mapsto LL1RAM.publicState,$
$\quad\quad privateStateEnc \mapsto LL1RAM.privateStateEnc,$
$\quad\quad historySummary \mapsto LL1RAM.historySummary,$
$\quad\quad authenticator \mapsto LL1RAM.authenticator]$

$\wedge \quad LL1RAM' \in LL1UntrustedStorageType \Rightarrow$
$\quad LL1RAM' = [$
$\quad\quad publicState \mapsto LL1RAM.publicState',$
$\quad\quad privateStateEnc \mapsto LL1RAM.privateStateEnc',$
$\quad\quad historySummary \mapsto LL1RAM.historySummary',$
$\quad\quad authenticator \mapsto LL1RAM.authenticator']$

⟨1⟩1. $LL1RAM \in LL1UntrustedStorageType \Rightarrow$
$\quad LL1RAM = [$
$\quad\quad publicState \mapsto LL1RAM.publicState,$
$\quad\quad privateStateEnc \mapsto LL1RAM.privateStateEnc,$
$\quad\quad historySummary \mapsto LL1RAM.historySummary,$
$\quad\quad authenticator \mapsto LL1RAM.authenticator]$

⟨2⟩1. HAVE $LL1RAM \in LL1UntrustedStorageType$

⟨2⟩ $ll1ram \triangleq [$
$\quad publicState \mapsto LL1RAM.publicState,$
$\quad privateStateEnc \mapsto LL1RAM.privateStateEnc,$
$\quad historySummary \mapsto LL1RAM.historySummary,$
$\quad authenticator \mapsto LL1RAM.authenticator]$

⟨2⟩2. $ll1ram =$
$\quad [i \in \{\text{"publicState"},$
$\quad\quad \text{"privateStateEnc"},$
$\quad\quad \text{"historySummary"},$
$\quad\quad \text{"authenticator"}\}$
$\quad\quad \mapsto ll1ram[i]]$

OBVIOUS

⟨2⟩3. $LL1RAM =$
$\quad [i \in \{\text{"publicState"},$
$\quad\quad \text{"privateStateEnc"},$
$\quad\quad \text{"historySummary"},$
$\quad\quad \text{"authenticator"}\}$

$\mapsto LL1RAM[i]]$

BY $\langle 2\rangle 1$ DEF $LL1UntrustedStorageType$

$\langle 2\rangle 4.$ QED

BY $\langle 2\rangle 2$, $\langle 2\rangle 3$

$\langle 1\rangle 2.$ $LL1RAM' \in LL1UntrustedStorageType \Rightarrow$
$LL1RAM' = [$
$publicState \mapsto LL1RAM.publicState'$,
$privateStateEnc \mapsto LL1RAM.privateStateEnc'$,
$historySummary \mapsto LL1RAM.historySummary'$,
$authenticator \mapsto LL1RAM.authenticator']$

$\langle 2\rangle 1.$ HAVE $LL1RAM' \in LL1UntrustedStorageType$

$\langle 2\rangle$ $ll1ramprime \triangleq [$
$publicState \mapsto LL1RAM.publicState'$,
$privateStateEnc \mapsto LL1RAM.privateStateEnc'$,
$historySummary \mapsto LL1RAM.historySummary'$,
$authenticator \mapsto LL1RAM.authenticator']$

$\langle 2\rangle 2.$ $ll1ramprime =$
$[i \in \{$"publicState",
"privateStateEnc",
"historySummary",
"authenticator"$\}$
$\mapsto ll1ramprime[i]]$

OBVIOUS

$\langle 2\rangle 3.$ $LL1RAM' =$
$[i \in \{$"publicState",
"privateStateEnc",
"historySummary",
"authenticator"$\}$
$\mapsto LL1RAM[i]']$

BY $\langle 2\rangle 1$ DEF $LL1UntrustedStorageType$

$\langle 2\rangle 4.$ QED

BY $\langle 2\rangle 2$, $\langle 2\rangle 3$

$\langle 1\rangle 3.$ QED

BY $\langle 1\rangle 1$, $\langle 1\rangle 2$

The $LL1NVRAMRecordCompositionLemma$ is a formality needed for the prover to compose an $LL1NVRAM$ record from fields of the appropriate type.

THEOREM $LL1NVRAMRecordCompositionLemma \triangleq$
$\land$ $LL1NVRAM \in LL1TrustedStorageType \Rightarrow$
$LL1NVRAM = [$
$historySummary \mapsto LL1NVRAM.historySummary$,
$symmetricKey \mapsto LL1NVRAM.symmetricKey]$
$\land$ $LL1NVRAM' \in LL1TrustedStorageType \Rightarrow$
$LL1NVRAM' = [$
$historySummary \mapsto LL1NVRAM.historySummary'$,
$symmetricKey \mapsto LL1NVRAM.symmetricKey']$

$\langle 1\rangle 1.$ $LL1NVRAM \in LL1TrustedStorageType \Rightarrow$
$LL1NVRAM = [$
$historySummary \mapsto LL1NVRAM.historySummary$,
$symmetricKey \mapsto LL1NVRAM.symmetricKey]$

$\langle 2 \rangle 1.$ HAVE $LL1NVRAM \in LL1TrustedStorageType$
$\langle 2 \rangle$ $ll1nvram \triangleq [$
$historySummary \mapsto LL1NVRAM.historySummary,$
$symmetricKey \mapsto LL1NVRAM.symmetricKey]$
$\langle 2 \rangle 2.$ $ll1nvram = [i \in \{\text{"historySummary"}, \text{"symmetricKey"}\} \mapsto ll1nvram[i]]$
OBVIOUS
$\langle 2 \rangle 3.$ $LL1NVRAM = [i \in \{\text{"historySummary"}, \text{"symmetricKey"}\} \mapsto LL1NVRAM[i]]$
BY $\langle 2 \rangle 1$ DEF $LL1TrustedStorageType$
$\langle 2 \rangle 4.$ QED
BY $\langle 2 \rangle 2$, $\langle 2 \rangle 3$
$\langle 1 \rangle 2.$ $LL1NVRAM' \in LL1TrustedStorageType \Rightarrow$
$LL1NVRAM' = [$
$historySummary \mapsto LL1NVRAM.historySummary',$
$symmetricKey \mapsto LL1NVRAM.symmetricKey']$
$\langle 2 \rangle 1.$ HAVE $LL1NVRAM' \in LL1TrustedStorageType$
$\langle 2 \rangle$ $ll1nvramprime \triangleq [$
$historySummary \mapsto LL1NVRAM.historySummary',$
$symmetricKey \mapsto LL1NVRAM.symmetricKey']$
$\langle 2 \rangle 2.$ $ll1nvramprime = [i \in \{\text{"historySummary"}, \text{"symmetricKey"}\} \mapsto ll1nvramprime[i]]$
OBVIOUS
$\langle 2 \rangle 3.$ $LL1NVRAM' = [i \in \{\text{"historySummary"}, \text{"symmetricKey"}\} \mapsto LL1NVRAM[i]']$
BY $\langle 2 \rangle 1$ DEF $LL1TrustedStorageType$
$\langle 2 \rangle 4.$ QED
BY $\langle 2 \rangle 2$, $\langle 2 \rangle 3$
$\langle 1 \rangle 3.$ QED
BY $\langle 1 \rangle 1$, $\langle 1 \rangle 2$

The *CheckpointHasBaseExtensionLemma* proves that a history summary produced by the *Checkpoint* function has an extension field that equals the base hash value.

THEOREM $CheckpointHasBaseExtensionLemma \triangleq$
$\forall historySummary \in HistorySummaryType :$
$Checkpoint(historySummary).extension = BaseHashValue$
$\langle 1 \rangle 1.$ TAKE $historySummary \in HistorySummaryType$
$\langle 1 \rangle$ $checkpointedAnchor \triangleq Hash(historySummary.anchor, historySummary.extension)$
$\langle 1 \rangle$ $checkpointedHistorySummary \triangleq [$
$anchor \mapsto checkpointedAnchor,$
$extension \mapsto BaseHashValue]$
$\langle 1 \rangle$ HIDE DEF $checkpointedAnchor$, $checkpointedHistorySummary$
$\langle 1 \rangle 2.$ CASE $historySummary.extension = BaseHashValue$
$\langle 2 \rangle 1.$ $Checkpoint(historySummary) = historySummary$
BY $\langle 1 \rangle 2$ DEF $Checkpoint$
$\langle 2 \rangle 2.$ QED
BY $\langle 1 \rangle 2$, $\langle 2 \rangle 1$
$\langle 1 \rangle 3.$ CASE $historySummary.extension \neq BaseHashValue$
$\langle 2 \rangle 1.$ $Checkpoint(historySummary) = checkpointedHistorySummary$
BY $\langle 1 \rangle 3$ DEF $Checkpoint$, $checkpointedHistorySummary$, $checkpointedAnchor$
$\langle 2 \rangle 2.$ $checkpointedHistorySummary.extension = BaseHashValue$
BY DEF $checkpointedHistorySummary$
$\langle 2 \rangle 3.$ QED
BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$

$\langle 1 \rangle 4.$ QED
BY $\langle 1 \rangle 2$, $\langle 1 \rangle 3$

The *SuccessorHasNonBaseExtensionLemma* proves that a history summary produced by the *Successor* function has an extension field that does not equal the base hash value.

THEOREM $\mathit{SuccessorHasNonBaseExtensionLemma} \triangleq$
$\forall\, \mathit{historySummary} \in \mathit{HistorySummaryType},\ \mathit{input} \in \mathit{InputType},\ \mathit{hashBarrier} \in \mathit{HashType} :$
$\mathit{Successor}(\mathit{historySummary},\ \mathit{input},\ \mathit{hashBarrier}).\mathit{extension} \neq \mathit{BaseHashValue}$
$\langle 1 \rangle 1.$ TAKE $\mathit{historySummary} \in \mathit{HistorySummaryType},\ \mathit{input} \in \mathit{InputType},\ \mathit{hashBarrier} \in \mathit{HashType}$
$\langle 1 \rangle\ \mathit{securedInput} \triangleq \mathit{Hash}(\mathit{hashBarrier},\ \mathit{input})$
$\langle 1 \rangle\ \mathit{newAnchor} \triangleq \mathit{historySummary}.\mathit{anchor}$
$\langle 1 \rangle\ \mathit{newExtension} \triangleq \mathit{Hash}(\mathit{historySummary}.\mathit{extension},\ \mathit{securedInput})$
$\langle 1 \rangle\ \mathit{newHistorySummary} \triangleq [$
$\mathit{anchor} \mapsto \mathit{newAnchor},$
$\mathit{extension} \mapsto \mathit{newExtension}]$
$\langle 1 \rangle$ HIDE DEF $\mathit{securedInput},\ \mathit{newAnchor},\ \mathit{newExtension},\ \mathit{newHistorySummary}$
$\langle 1 \rangle 2.\ \mathit{newExtension} \neq \mathit{BaseHashValue}$
$\langle 2 \rangle 1.\ \mathit{historySummary}.\mathit{extension} \in \mathit{HashDomain}$
$\langle 3 \rangle 1.\ \mathit{historySummary}.\mathit{extension} \in \mathit{HashType}$
$\langle 4 \rangle 1.\ \mathit{historySummary} \in \mathit{HistorySummaryType}$
BY $\langle 1 \rangle 1$
$\langle 4 \rangle 2.$ QED
BY $\langle 4 \rangle 1$ DEF $\mathit{HistorySummaryType}$
$\langle 3 \rangle 2.$ QED
BY $\langle 3 \rangle 1$ DEF $\mathit{HashDomain}$
$\langle 2 \rangle 2.\ \mathit{securedInput} \in \mathit{HashDomain}$
$\langle 3 \rangle 1.\ \mathit{securedInput} \in \mathit{HashType}$
$\langle 4 \rangle 1.\ \mathit{hashBarrier} \in \mathit{HashDomain}$
$\langle 5 \rangle 1.\ \mathit{hashBarrier} \in \mathit{HashType}$
BY $\langle 1 \rangle 1$
$\langle 5 \rangle 2.$ QED
BY $\langle 5 \rangle 1$ DEF $\mathit{HashDomain}$
$\langle 4 \rangle 2.\ \mathit{input} \in \mathit{HashDomain}$
$\langle 5 \rangle 1.\ \mathit{input} \in \mathit{InputType}$
BY $\langle 1 \rangle 1$
$\langle 5 \rangle 2.$ QED
BY $\langle 5 \rangle 1$ DEF $\mathit{HashDomain}$
$\langle 4 \rangle 3.$ QED
BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$, $\mathit{HashTypeSafe}$DEF $\mathit{securedInput}$
$\langle 3 \rangle 2.$ QED
BY $\langle 3 \rangle 1$ DEF $\mathit{HashDomain}$
$\langle 2 \rangle 3.$ QED
BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, $\mathit{BaseHashValueUnique}$DEF $\mathit{newExtension}$
$\langle 1 \rangle 3.\ \mathit{Successor}(\mathit{historySummary},\ \mathit{input},\ \mathit{hashBarrier}).\mathit{extension} = \mathit{newExtension}$
BY DEF $\mathit{Successor},\ \mathit{newExtension},\ \mathit{securedInput}$
$\langle 1 \rangle 4.$ QED
BY $\langle 1 \rangle 2$, $\langle 1 \rangle 3$

The *HistorySummariesMatchUniqueLemma* asserts that there is a unique Memoir-Basic history summary that matches a particular Memoir-Opt history summary.

THEOREM $HistorySummariesMatchUniqueLemma \triangleq$
$\forall\ ll1HistorySummary1,\ ll1HistorySummary2 \in HashType,$
$ll2HistorySummary \in HistorySummaryType,$
$hashBarrier \in HashType :$
$(\wedge\ HistorySummariesMatch(ll1HistorySummary1,\ ll2HistorySummary,\ hashBarrier)$
$\wedge\ HistorySummariesMatch(ll1HistorySummary2,\ ll2HistorySummary,\ hashBarrier))$
$\Rightarrow$
$ll1HistorySummary1 = ll1HistorySummary2$

It is convenient to define a predicate that captures the quantified expression of the theorem.

$\langle 1 \rangle\ HistorySummariesMatchUnique($
$ll1HistorySummary1,\ ll1HistorySummary2,\ ll2HistorySummary,\ hashBarrier) \triangleq$
$(\wedge\ HistorySummariesMatch(ll1HistorySummary1,\ ll2HistorySummary,\ hashBarrier)$
$\wedge\ HistorySummariesMatch(ll1HistorySummary2,\ ll2HistorySummary,\ hashBarrier))$
$\Rightarrow$
$ll1HistorySummary1 = ll1HistorySummary2$

To prove the universally quantified expressoin, we take a set of variables of the appropriate types.

$\langle 1 \rangle 1.$ TAKE $ll1HistorySummary1,\ ll1HistorySummary2 \in HashType,$
$ll2HistorySummary \in HistorySummaryType,$
$hashBarrier \in HashType$

$\langle 1 \rangle\ ll2InitialHistorySummary \triangleq [anchor \mapsto BaseHashValue,\ extension \mapsto BaseHashValue]$

Our proof is inductive. First, we prove the base case, which is when the Memoir-Opt history summary equals the initial history summary.

$\langle 1 \rangle 2.\ ll2HistorySummary = ll2InitialHistorySummary \Rightarrow$
$HistorySummariesMatchUnique($
$ll1HistorySummary1,\ ll1HistorySummary2,\ ll2HistorySummary,\ hashBarrier)$

We assume the antecedent of the base step.

$\langle 2 \rangle 1.$ HAVE $ll2HistorySummary = ll2InitialHistorySummary$

The *HistorySummariesMatchUnique* predicate states an implication. It suffices to assume the antecedent and prove the consequent.

$\langle 2 \rangle 2.$ SUFFICES
ASSUME
$\wedge\ HistorySummariesMatch(ll1HistorySummary1,\ ll2HistorySummary,\ hashBarrier)$
$\wedge\ HistorySummariesMatch(ll1HistorySummary2,\ ll2HistorySummary,\ hashBarrier)$
PROVE
$ll1HistorySummary1 = ll1HistorySummary2$
BY DEF $HistorySummariesMatchUnique$

Memoir-Basic history summary 1 equals the base hash value, by the definition of *HistorySummariesMatch* for a Memoir-Opt history summary that equals the initial history summary.

$\langle 2 \rangle 3.\ ll1HistorySummary1 = BaseHashValue$
$\langle 3 \rangle 1.\ HistorySummariesMatch(ll1HistorySummary1,\ ll2InitialHistorySummary,\ hashBarrier) =$
$(ll1HistorySummary1 = BaseHashValue)$
$\langle 4 \rangle 1.\ ll2InitialHistorySummary \in HistorySummaryType$
BY $\langle 1 \rangle 1$, $\langle 2 \rangle 1$
$\langle 4 \rangle 2.$ QED
BY $\langle 1 \rangle 1$, $\langle 4 \rangle 1$, $HistorySummariesMatchDefinition$
$\langle 3 \rangle 2.\ HistorySummariesMatch(ll1HistorySummary1,\ ll2InitialHistorySummary,\ hashBarrier)$
BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$
$\langle 3 \rangle 3.$ QED

BY ⟨3⟩1, ⟨3⟩2

Memoir-Basic history summary 2 equals the base hash value, by the definition of *HistorySummariesMatch* for a Memoir-Opt history summary that equals the initial history summary.

⟨2⟩4. $ll1HistorySummary2 = BaseHashValue$

⟨3⟩1. $HistorySummariesMatch(ll1HistorySummary2, ll2InitialHistorySummary, hashBarrier) = (ll1HistorySummary2 = BaseHashValue)$

⟨4⟩1. $ll2InitialHistorySummary \in HistorySummaryType$

BY ⟨1⟩1, ⟨2⟩1

⟨4⟩2. QED

BY ⟨1⟩1, ⟨4⟩1, *HistorySummariesMatchDefinition*

⟨3⟩2. $HistorySummariesMatch(ll1HistorySummary2, ll2InitialHistorySummary, hashBarrier)$

BY ⟨2⟩1, ⟨2⟩2

⟨3⟩3. QED

BY ⟨3⟩1, ⟨3⟩2

Since the two Memoir-Basic history summaries each equal the base hash value, they equal each other.

⟨2⟩5. QED

BY ⟨2⟩3, ⟨2⟩4

Second, we prove the inductive case: If the previous history summaries defined in the *HistorySummariesMatchRecursion* predicate are unique, then their successor history summaries are unique.

⟨1⟩3. $\forall$ $input1, input2 \in InputType,$
$previousLL1HistorySummary1, previousLL1HistorySummary2 \in HashType,$
$previousLL2HistorySummary \in HistorySummaryType :$

$$
\begin{aligned}
&(\land\ HistorySummariesMatchRecursion(\\
&\qquad ll1HistorySummary1, ll2HistorySummary, hashBarrier)!(\\
&\qquad input1, previousLL1HistorySummary1, previousLL2HistorySummary)\\
&\ \ \land\ HistorySummariesMatchRecursion(\\
&\qquad ll1HistorySummary2, ll2HistorySummary, hashBarrier)!(\\
&\qquad input2, previousLL1HistorySummary2, previousLL2HistorySummary)\\
&\ \ \land\ HistorySummariesMatchUnique(\\
&\qquad previousLL1HistorySummary1,\\
&\qquad previousLL1HistorySummary2,\\
&\qquad previousLL2HistorySummary,\\
&\qquad hashBarrier))\\
&\Rightarrow\\
&\quad HistorySummariesMatchUnique(\\
&\qquad ll1HistorySummary1, ll1HistorySummary2, ll2HistorySummary, hashBarrier)
\end{aligned}
$$

To prove the universally quantified expressoin, we take a set of variables of the appropriate types.

⟨2⟩1. TAKE $input1, input2 \in InputType,$
$previousLL1HistorySummary1, previousLL1HistorySummary2 \in HashType,$
$previousLL2HistorySummary \in HistorySummaryType$

We assume the antecedent of the inductive step.

⟨2⟩2. HAVE

$$
\begin{aligned}
&\land\ HistorySummariesMatchRecursion(\\
&\qquad ll1HistorySummary1, ll2HistorySummary, hashBarrier)!(\\
&\qquad input1, previousLL1HistorySummary1, previousLL2HistorySummary)\\
&\land\ HistorySummariesMatchRecursion(\\
&\qquad ll1HistorySummary2, ll2HistorySummary, hashBarrier)!(\\
&\qquad input2, previousLL1HistorySummary2, previousLL2HistorySummary)\\
&\land\ HistorySummariesMatchUnique(\\
&\qquad previousLL1HistorySummary1,\\
&\qquad previousLL1HistorySummary2,\\
&\qquad previousLL2HistorySummary,
\end{aligned}
$$

$hashBarrier)$

The *HistorySummariesMatchUnique* predicate states an implication. It suffices to assume the antecedent and prove the consequent.

$\langle 2 \rangle 3.$ SUFFICES
ASSUME
$\land$ $HistorySummariesMatch(ll1HistorySummary1, ll2HistorySummary, hashBarrier)$
$\land$ $HistorySummariesMatch(ll1HistorySummary2, ll2HistorySummary, hashBarrier)$
PROVE
$ll1HistorySummary1 = ll1HistorySummary2$
BY DEF $HistorySummariesMatchUnique$

We prove that the two inputs from the separate instances of the *HistorySummariesMatchRecursion* predicate are equal to each other.

$\langle 2 \rangle 4.$ $input1 = input2$
$\langle 3 \rangle 1.$ $LL2HistorySummaryIsSuccessor($
$ll2HistorySummary, previousLL2HistorySummary, input1, hashBarrier)$
BY $\langle 2 \rangle 2$

We re-state the definitions from *LL2HistorySummaryIsSuccessor* for input 1.

$\langle 3 \rangle$ $ll2SuccessorHistorySummary1 \triangleq Successor(previousLL2HistorySummary, input1, hashBarrier)$
$\langle 3 \rangle$ $ll2CheckpointedSuccessorHistorySummary1 \triangleq Checkpoint(ll2SuccessorHistorySummary1)$

We hide the definitions.

$\langle 3 \rangle$ HIDE DEF $ll2SuccessorHistorySummary1, ll2CheckpointedSuccessorHistorySummary1$

We re-state the definitions from *Successor* for input 1.

$\langle 3 \rangle$ $securedInput1 \triangleq Hash(hashBarrier, input1)$
$\langle 3 \rangle$ $newAnchor \triangleq previousLL2HistorySummary.anchor$
$\langle 3 \rangle$ $newExtension1 \triangleq Hash(previousLL2HistorySummary.extension, securedInput1)$
$\langle 3 \rangle$ $ll2NewHistorySummary1 \triangleq [$
$anchor \mapsto newAnchor,$
$extension \mapsto newExtension1]$

We prove the types of the definitions from *Successor* and the definition of *ll2SuccessorHistorySummary1*, with help from the *SuccessorDefsTypeSafeLemma*.

$\langle 3 \rangle 2.$ $\land$ $securedInput1 \in HashType$
$\land$ $newAnchor \in HashType$
$\land$ $newExtension1 \in HashType$
$\land$ $ll2NewHistorySummary1 \in HistorySummaryType$
$\land$ $ll2NewHistorySummary1.anchor \in HashType$
$\land$ $ll2NewHistorySummary1.extension \in HashType$
BY $\langle 1 \rangle 1$, $\langle 2 \rangle 1$, $SuccessorDefsTypeSafeLemma$
$\langle 3 \rangle 3.$ $ll2SuccessorHistorySummary1 \in HistorySummaryType$
BY $\langle 3 \rangle 2$ DEF $ll2SuccessorHistorySummary1, Successor$

We hide the definitions.

$\langle 3 \rangle$ HIDE DEF $securedInput1, newAnchor, newExtension1, ll2NewHistorySummary1$

We re-state the definitions from *Checkpoint* for input 1.

$\langle 3 \rangle$ $checkpointedAnchor1 \triangleq$
$Hash(ll2SuccessorHistorySummary1.anchor, ll2SuccessorHistorySummary1.extension)$
$\langle 3 \rangle$ $ll2CheckpointedHistorySummary1 \triangleq [$
$anchor \mapsto checkpointedAnchor1,$
$extension \mapsto BaseHashValue]$

We prove the types of the definitions from *Checkpoint* and the definition of *ll2CheckpointedSuccessorHistorySummary1*, with help from the *CheckpointDefsTypeSafeLemma*.

$\langle 3 \rangle 4.$ $\land$ $checkpointedAnchor1 \in HashType$

$\land$ $ll2CheckpointedHistorySummary1 \in HistorySummaryType$
$\land$ $ll2CheckpointedHistorySummary1.anchor \in HashType$
$\land$ $ll2CheckpointedHistorySummary1.extension \in HashType$
BY $\langle 3 \rangle 3$, $CheckpointDefsTypeSafeLemma$
$\langle 3 \rangle 5$. $ll2CheckpointedSuccessorHistorySummary1 \in HistorySummaryType$
BY $\langle 3 \rangle 3$, $\langle 3 \rangle 4$ DEF $ll2CheckpointedSuccessorHistorySummary1$, $Checkpoint$

We hide the definitions.

$\langle 3 \rangle$ HIDE DEF $checkpointedAnchor1$, $ll2CheckpointedHistorySummary1$
$\langle 3 \rangle 6$. $LL2HistorySummaryIsSuccessor($
$ll2HistorySummary, previousLL2HistorySummary, input2, hashBarrier)$
BY $\langle 2 \rangle 2$

We re-state the definitions from *LL2HistorySummaryIsSuccessor* for input 2.

$\langle 3 \rangle$ $ll2SuccessorHistorySummary2 \triangleq Successor(previousLL2HistorySummary, input2, hashBarrier)$
$\langle 3 \rangle$ $ll2CheckpointedSuccessorHistorySummary2 \triangleq Checkpoint(ll2SuccessorHistorySummary2)$

We hide the definitions.

$\langle 3 \rangle$ HIDE DEF $ll2SuccessorHistorySummary2$, $ll2CheckpointedSuccessorHistorySummary2$

We re-state the definitions from *Successor* for input 2.

$\langle 3 \rangle$ $securedInput2 \triangleq Hash(hashBarrier, input2)$
$\langle 3 \rangle$ $newExtension2 \triangleq Hash(previousLL2HistorySummary.extension, securedInput2)$
$\langle 3 \rangle$ $ll2NewHistorySummary2 \triangleq [$
$anchor \mapsto newAnchor,$
$extension \mapsto newExtension2]$

We prove the types of the definitions from *Successor* and the definition of *ll2SuccessorHistorySummary2*, with help from the *SuccessorDefsTypeSafeLemma*.

$\langle 3 \rangle 7$. $\land$ $securedInput2 \in HashType$
$\land$ $newExtension2 \in HashType$
$\land$ $ll2NewHistorySummary2 \in HistorySummaryType$
$\land$ $ll2NewHistorySummary2.anchor \in HashType$
$\land$ $ll2NewHistorySummary2.extension \in HashType$
BY $\langle 1 \rangle 1$, $\langle 2 \rangle 1$, $SuccessorDefsTypeSafeLemma$DEF $newAnchor$
$\langle 3 \rangle 8$. $ll2SuccessorHistorySummary2 \in HistorySummaryType$
BY $\langle 3 \rangle 7$ DEF $ll2SuccessorHistorySummary2$, $Successor$, $newAnchor$

We hide the definitions.

$\langle 3 \rangle$ HIDE DEF $securedInput2$, $newExtension2$, $ll2NewHistorySummary2$

We re-state the definitions from *Checkpoint* for input 2.

$\langle 3 \rangle$ $checkpointedAnchor2 \triangleq$
$Hash(ll2SuccessorHistorySummary2.anchor, ll2SuccessorHistorySummary2.extension)$
$\langle 3 \rangle$ $ll2CheckpointedHistorySummary2 \triangleq [$
$anchor \mapsto checkpointedAnchor2,$
$extension \mapsto BaseHashValue]$

We prove the types of the definitions from *Checkpoint* and the definition of *ll2CheckpointedSuccessorHistorySummary2*, with help from the *CheckpointDefsTypeSafeLemma*.

$\langle 3 \rangle 9$. $\land$ $checkpointedAnchor2 \in HashType$
$\land$ $ll2CheckpointedHistorySummary2 \in HistorySummaryType$
$\land$ $ll2CheckpointedHistorySummary2.anchor \in HashType$
$\land$ $ll2CheckpointedHistorySummary2.extension \in HashType$
BY $\langle 3 \rangle 8$, $CheckpointDefsTypeSafeLemma$
$\langle 3 \rangle 10$. $ll2CheckpointedSuccessorHistorySummary2 \in HistorySummaryType$
BY $\langle 3 \rangle 8$, $\langle 3 \rangle 9$ DEF $ll2CheckpointedSuccessorHistorySummary2$, $Checkpoint$

We hide the definitions.

$\langle 3 \rangle$ HIDE DEF *checkpointedAnchor*2, *ll2CheckpointedHistorySummary*2

We prove that the successor history summaries, as defined in the LET of the *LL2HistorySummaryIsSuccessor* predicate, are equal to each other.

$\langle 3 \rangle$11. *ll2SuccessorHistorySummary*1 = *ll2SuccessorHistorySummary*2

We consider two cases. In the first case, the Memoir-Opt history summary has an extension field that equals the base hash value.

$\langle 4 \rangle$1. CASE *ll2HistorySummary.extension* = *BaseHashValue*

The checkpointed successor history summaries, as defined in the LET within the *LL2HistorySummaryIsSuccessor* predicate, are equal.

$\langle 5 \rangle$1. *ll2CheckpointedSuccessorHistorySummary*1 = *ll2CheckpointedSuccessorHistorySummary*2

The *LL2HistorySummaryIsSuccessor* predicate expresses a disjunction. An history summary can be a successor either by being a direct successor or by being a checkpoint of a successor. We prove that, in this case, the history summary is a checkpoint of a successor formed with input 1. It cannot be a direct successor because its extension field is the base hash value, and any direct successor will have a non-base extension field.

$\langle 6 \rangle$1. *ll2HistorySummary* = *ll2CheckpointedSuccessorHistorySummary*1
$\langle 7 \rangle$1. $\lor$ *ll2HistorySummary* = *ll2SuccessorHistorySummary*1
$\lor$ *ll2HistorySummary* = *ll2CheckpointedSuccessorHistorySummary*1
$\langle 8 \rangle$1. *LL2HistorySummaryIsSuccessor*(
ll2HistorySummary, *previousLL2HistorySummary*, *input*1, *hashBarrier*)
BY $\langle 2 \rangle$2
$\langle 8 \rangle$2. QED
BY $\langle 8 \rangle$1
DEF *LL2HistorySummaryIsSuccessor*, *ll2SuccessorHistorySummary*1,
*ll2CheckpointedSuccessorHistorySummary*1
$\langle 7 \rangle$2. *ll2HistorySummary* $\neq$ *ll2SuccessorHistorySummary*1
$\langle 8 \rangle$1. *ll2SuccessorHistorySummary*1*.extension* $\neq$ *BaseHashValue*
BY $\langle 1 \rangle$1, $\langle 2 \rangle$1, *SuccessorHasNonBaseExtensionLemma*DEF *ll2SuccessorHistorySummary*1
$\langle 8 \rangle$2. QED
BY $\langle 4 \rangle$1, $\langle 8 \rangle$1
$\langle 7 \rangle$3. QED
BY $\langle 7 \rangle$1, $\langle 7 \rangle$2

The same logic as above applies to the the checkpointed successor formed with input 2.

$\langle 6 \rangle$2. *ll2HistorySummary* = *ll2CheckpointedSuccessorHistorySummary*2
$\langle 7 \rangle$1. $\lor$ *ll2HistorySummary* = *ll2SuccessorHistorySummary*2
$\lor$ *ll2HistorySummary* = *ll2CheckpointedSuccessorHistorySummary*2
$\langle 8 \rangle$1. *LL2HistorySummaryIsSuccessor*(
ll2HistorySummary, *previousLL2HistorySummary*, *input*2, *hashBarrier*)
BY $\langle 2 \rangle$2
$\langle 8 \rangle$2. QED
BY $\langle 8 \rangle$1
DEF *LL2HistorySummaryIsSuccessor*, *ll2SuccessorHistorySummary*2,
*ll2CheckpointedSuccessorHistorySummary*2
$\langle 7 \rangle$2. *ll2HistorySummary* $\neq$ *ll2SuccessorHistorySummary*2
$\langle 8 \rangle$1. *ll2SuccessorHistorySummary*2*.extension* $\neq$ *BaseHashValue*
BY $\langle 1 \rangle$1, $\langle 2 \rangle$1, *SuccessorHasNonBaseExtensionLemma*DEF *ll2SuccessorHistorySummary*2
$\langle 8 \rangle$2. QED
BY $\langle 4 \rangle$1, $\langle 8 \rangle$1
$\langle 7 \rangle$3. QED
BY $\langle 7 \rangle$1, $\langle 7 \rangle$2

Because the two checkpointed successor history summaries are each equal to the same value, they are equal to each other.

$\langle 6 \rangle 3.$ QED

BY $\langle 6 \rangle 1$, $\langle 6 \rangle 2$

We prove the conclusion: The successor history summaries, as defined in the LET of the *LL2HistorySummaryIsSuccessor* predicate, are equal to each other.

$\langle 5 \rangle 2.$ QED

The checkpointed history summaries, as defined in the LET of the *Checkpoint* operator, are equal to each other.

$\langle 6 \rangle 1.$ $ll2CheckpointedHistorySummary1 = ll2CheckpointedHistorySummary2$

As proven above, the checkpointed history summaries, as defined in the LET of the *LL2HistorySummaryIsSuccessor* predicate, are equal.

$\langle 7 \rangle 1.$ $ll2CheckpointedSuccessorHistorySummary1 = ll2CheckpointedSuccessorHistorySummary2$

BY $\langle 5 \rangle 1$

The extension field of each successor history summary is not equal to the base hash value, because they are direct successors.

$\langle 7 \rangle 2.$ $ll2SuccessorHistorySummary1.extension \neq BaseHashValue$

BY $\langle 1 \rangle 1$, $\langle 2 \rangle 1$, *SuccessorHasNonBaseExtensionLemma*DEF *ll2SuccessorHistorySummary1*

$\langle 7 \rangle 3.$ $ll2SuccessorHistorySummary2.extension \neq BaseHashValue$

BY $\langle 1 \rangle 1$, $\langle 2 \rangle 2$, *SuccessorHasNonBaseExtensionLemma*DEF *ll2SuccessorHistorySummary2*

The conclusion follows from the definitions, because the Checkpoint operator is injective under the preimage constraint of an extension field that is unequal to the base hash value.

$\langle 7 \rangle 4.$ QED

BY $\langle 7 \rangle 1$, $\langle 7 \rangle 2$, $\langle 7 \rangle 3$

DEF *ll2CheckpointedSuccessorHistorySummary1*, *ll2CheckpointedSuccessorHistorySummary2*, *Checkpoint*, *ll2CheckpointedHistorySummary1*, *ll2CheckpointedHistorySummary2*, *checkpointedAnchor1*, *checkpointedAnchor2*

The individual fields of the successor history summaries are equal to each other, thanks to the collision resistance of the hash function.

$\langle 6 \rangle 2.$ $\wedge$ $ll2SuccessorHistorySummary1.anchor = ll2SuccessorHistorySummary2.anchor$

$\wedge$ $ll2SuccessorHistorySummary1.extension = ll2SuccessorHistorySummary2.extension$

$\langle 7 \rangle 1.$ $checkpointedAnchor1 = checkpointedAnchor2$

BY $\langle 6 \rangle 1$ DEF *ll2CheckpointedHistorySummary1*, *ll2CheckpointedHistorySummary2*

$\langle 7 \rangle 2.$ $ll2SuccessorHistorySummary1.anchor \in HashDomain$

BY $\langle 3 \rangle 3$ DEF *HistorySummaryType*, *HashDomain*

$\langle 7 \rangle 3.$ $ll2SuccessorHistorySummary1.extension \in HashDomain$

BY $\langle 3 \rangle 3$ DEF *HistorySummaryType*, *HashDomain*

$\langle 7 \rangle 4.$ $ll2SuccessorHistorySummary2.anchor \in HashDomain$

BY $\langle 3 \rangle 8$ DEF *HistorySummaryType*, *HashDomain*

$\langle 7 \rangle 5.$ $ll2SuccessorHistorySummary2.extension \in HashDomain$

BY $\langle 3 \rangle 8$ DEF *HistorySummaryType*, *HashDomain*

$\langle 7 \rangle 6.$ QED

Ideally, this QED step should just read:

BY $\langle 7 \rangle 1$, $\langle 7 \rangle 2$, $\langle 7 \rangle 3$, $\langle 7 \rangle 4$, $\langle 7 \rangle 5$, *HashCollisionResistant* DEF *checkpointedAnchor1*, *checkpointedAnchor2*

However, the prover seems to get a little confused in this instance. We make life easier for the prover by definining some local variables and hiding their definitions before appealing to the *HashCollisionResistant* assumption.

$\langle 8 \rangle$ $h1a \triangleq ll2SuccessorHistorySummary1.anchor$

$\langle 8 \rangle$ $h2a \triangleq ll2SuccessorHistorySummary1.extension$

$\langle 8 \rangle$ $h1b \triangleq ll2SuccessorHistorySummary2.anchor$

$\langle 8 \rangle$ $h2b \triangleq ll2SuccessorHistorySummary2.extension$

$\langle 8 \rangle 1.$ $h1a \in HashDomain$

BY $\langle 7 \rangle 2$

⟨8⟩2. $h2a \in HashDomain$
BY ⟨7⟩3
⟨8⟩3. $h1b \in HashDomain$
BY ⟨7⟩4
⟨8⟩4. $h2b \in HashDomain$
BY ⟨7⟩5
⟨8⟩5. $Hash(h1a, h2a) = Hash(h1b, h2b)$
BY ⟨7⟩1 DEF $checkpointedAnchor1$, $checkpointedAnchor2$
⟨8⟩6. $h1a = h1b \land h2a = h2b$
⟨9⟩ HIDE DEF $h1a$, $h2a$, $h1b$, $h2b$
⟨9⟩1. QED
BY ⟨8⟩1, ⟨8⟩2, ⟨8⟩3, ⟨8⟩4, ⟨8⟩5, $HashCollisionResistant$
⟨8⟩7. QED
BY ⟨8⟩6

Because the fields are equal, the records are equal, but proving this requires that we prove the types of the records and invoke the *HistorySummaryRecordCompositionLemma*.

⟨6⟩3. $ll2SuccessorHistorySummary1 \in HistorySummaryType$
BY ⟨3⟩3
⟨6⟩4. $ll2SuccessorHistorySummary2 \in HistorySummaryType$
BY ⟨3⟩8
⟨6⟩5. QED
BY ⟨6⟩2, ⟨6⟩3, ⟨6⟩4, $HistorySummaryRecordCompositionLemma$ DEF $HistorySummaryType$

In the second case, the Memoir-Opt history summary has an extension field that does not equal the base hash value.

⟨4⟩2. CASE $ll2HistorySummary.extension \neq BaseHashValue$

The *LL2HistorySummaryIsSuccessor* predicate expresses a disjunction. An history summary can be a successor either by being a direct successor or by being a checkpoint of a successor. We prove that, in this case, history summary 1 is a direct successor formed with input 1. It cannot be a checkpoint because its extension field is not the base hash value, and any checkpoint will have a base extension field.

⟨5⟩1. $ll2HistorySummary = ll2SuccessorHistorySummary1$
⟨6⟩1. $\lor\ ll2HistorySummary = ll2SuccessorHistorySummary1$
$\lor\ ll2HistorySummary = ll2CheckpointedSuccessorHistorySummary1$
⟨7⟩1. $LL2HistorySummaryIsSuccessor($
$ll2HistorySummary, previousLL2HistorySummary, input1, hashBarrier)$
BY ⟨2⟩2
⟨7⟩2. QED
BY ⟨7⟩1
DEF $LL2HistorySummaryIsSuccessor$, $ll2SuccessorHistorySummary1$,
$ll2CheckpointedSuccessorHistorySummary1$
⟨6⟩2. $ll2HistorySummary \neq ll2CheckpointedSuccessorHistorySummary1$
⟨7⟩1. $ll2CheckpointedSuccessorHistorySummary1.extension = BaseHashValue$
BY ⟨3⟩3, $CheckpointHasBaseExtensionLemma$
DEF $ll2CheckpointedSuccessorHistorySummary1$
⟨7⟩2. QED
BY ⟨4⟩2, ⟨7⟩1
⟨6⟩3. QED
BY ⟨6⟩1, ⟨6⟩2

The same logic as above applies to the the successor formed with input 2.

⟨5⟩2. $ll2HistorySummary = ll2SuccessorHistorySummary2$
⟨6⟩1. $\lor\ ll2HistorySummary = ll2SuccessorHistorySummary2$
$\lor\ ll2HistorySummary = ll2CheckpointedSuccessorHistorySummary2$
⟨7⟩1. $LL2HistorySummaryIsSuccessor($

$ll2HistorySummary$, $previousLL2HistorySummary$, $input2$, $hashBarrier$)
BY $\langle 2 \rangle 2$
$\langle 7 \rangle 2$. QED
BY $\langle 7 \rangle 1$
DEF $LL2HistorySummaryIsSuccessor$, $ll2SuccessorHistorySummary2$,
$ll2CheckpointedSuccessorHistorySummary2$
$\langle 6 \rangle 2$. $ll2HistorySummary \neq ll2CheckpointedSuccessorHistorySummary2$
$\langle 7 \rangle 1$. $ll2CheckpointedSuccessorHistorySummary2.extension = BaseHashValue$
BY $\langle 3 \rangle 8$, $CheckpointHasBaseExtensionLemma$
DEF $ll2CheckpointedSuccessorHistorySummary2$
$\langle 7 \rangle 2$. QED
BY $\langle 4 \rangle 2$, $\langle 7 \rangle 1$
$\langle 6 \rangle 3$. QED
BY $\langle 6 \rangle 1$, $\langle 6 \rangle 2$

Because the two successor history summaries are each equal to the same value, they are equal to each other.

$\langle 5 \rangle 3$. QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$

The two cases are exhaustive.

$\langle 4 \rangle 3$. QED
BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$

The secured inputss are equal to each other, because the new extensions are equal to each other, and the hash is collision-resistant.

$\langle 3 \rangle 12$. $securedInput1 = securedInput2$
$\langle 4 \rangle 1$. $newExtension1 = newExtension2$
$\langle 5 \rangle 1$. $ll2NewHistorySummary1 = ll2NewHistorySummary2$
BY $\langle 3 \rangle 11$
DEF $ll2SuccessorHistorySummary1$, $ll2SuccessorHistorySummary2$, $Successor$,
$ll2NewHistorySummary1$, $ll2NewHistorySummary2$, $newExtension1$, $newExtension2$,
$newAnchor$, $securedInput1$, $securedInput2$
$\langle 5 \rangle 2$. QED
BY $\langle 5 \rangle 1$ DEF $ll2NewHistorySummary1$, $ll2NewHistorySummary2$
$\langle 4 \rangle 2$. $previousLL2HistorySummary.extension \in HashDomain$
BY $\langle 2 \rangle 1$ DEF $HistorySummaryType$, $HashDomain$
$\langle 4 \rangle 3$. $securedInput1 \in HashDomain$
BY $\langle 3 \rangle 2$ DEF $HashDomain$
$\langle 4 \rangle 4$. $securedInput2 \in HashDomain$
BY $\langle 3 \rangle 7$ DEF $HashDomain$
$\langle 4 \rangle 5$. QED

Ideally, this QED step should just read:

BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$, $\langle 4 \rangle 3$, $\langle 4 \rangle 4$, $HashCollisionResistant$

However, the prover seems to get a little confused in this instance. We make life easier for the prover by definining some local variables and hiding their definitions before appealing to the *HashCollisionResistant* assumption.

$\langle 5 \rangle$ $h1a \triangleq previousLL2HistorySummary.extension$
$\langle 5 \rangle$ $h2a \triangleq securedInput1$
$\langle 5 \rangle$ $h2b \triangleq securedInput2$
$\langle 5 \rangle 1$. $h1a \in HashDomain$
BY $\langle 4 \rangle 2$
$\langle 5 \rangle 2$. $h2a \in HashDomain$
BY $\langle 4 \rangle 3$
$\langle 5 \rangle 3$. $h2b \in HashDomain$
BY $\langle 4 \rangle 4$

$\langle 5 \rangle 4.$ $Hash(h1a,\ h2a) = Hash(h1a,\ h2b)$
BY $\langle 4 \rangle 1$ DEF $newExtension1$, $newExtension2$
$\langle 5 \rangle 5.$ $h2a = h2b$
$\langle 6 \rangle$ HIDE DEF $h1a$, $h2a$, $h2b$
$\langle 6 \rangle 1.$ QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$, $\langle 5 \rangle 3$, $\langle 5 \rangle 4$, $HashCollisionResistant$
$\langle 5 \rangle 6.$ QED
BY $\langle 5 \rangle 5$

The input values are equal to each other, because the secured inputs are equal, and the hash function is collision-resistant.

$\langle 3 \rangle 16.$ QED
$\langle 4 \rangle 1.$ $securedInput1 = securedInput2$
BY $\langle 3 \rangle 12$
$\langle 4 \rangle 2.$ $hashBarrier \in HashDomain$
BY $\langle 1 \rangle 1$ DEF $HashDomain$
$\langle 4 \rangle 3.$ $input1 \in HashDomain$
BY $\langle 2 \rangle 1$ DEF $HashDomain$
$\langle 4 \rangle 4.$ $input2 \in HashDomain$
BY $\langle 2 \rangle 2$ DEF $HashDomain$
$\langle 4 \rangle 5.$ QED
BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$, $\langle 4 \rangle 3$, $\langle 4 \rangle 4$, $HashCollisionResistant$
DEF $securedInput1$, $securedInput2$

We prove that the two previous Memoir-Basic history summaries from the separate instances of the *HistorySummariesMatchRecursion* predicate are equal to each other. This follows from the recursive use of the *HistorySummariesMatchUnique* predicate on the previous history summaries.

$\langle 2 \rangle 5.$ $previousLL1HistorySummary1 = previousLL1HistorySummary2$
$\langle 3 \rangle 1.$ $HistorySummariesMatch($
$previousLL1HistorySummary1,\ previousLL2HistorySummary,\ hashBarrier)$
BY $\langle 2 \rangle 2$ DEF $HistorySummariesMatchRecursion$
$\langle 3 \rangle 2.$ $HistorySummariesMatch($
$previousLL1HistorySummary2,\ previousLL2HistorySummary,\ hashBarrier)$
BY $\langle 2 \rangle 2$ DEF $HistorySummariesMatchRecursion$
$\langle 3 \rangle 3.$ $HistorySummariesMatchUnique($
$previousLL1HistorySummary1,$
$previousLL1HistorySummary2,$
$previousLL2HistorySummary,$
$hashBarrier)$
BY $\langle 2 \rangle 2$
$\langle 3 \rangle 4.$ QED
BY $\langle 3 \rangle 1$, $\langle 3 \rangle 2$, $\langle 3 \rangle 3$ DEF $HistorySummariesMatchUnique$

The conclusion follows directly from the equality of the previous history summaries and the inputs, since the current history summaries are generated as hashes of those values.

$\langle 2 \rangle 6.$ QED
$\langle 3 \rangle 1.$ $ll1HistorySummary1 = Hash(previousLL1HistorySummary1,\ input1)$
BY $\langle 2 \rangle 2$ DEF $HistorySummariesMatchRecursion$
$\langle 3 \rangle 2.$ $ll1HistorySummary2 = Hash(previousLL1HistorySummary2,\ input2)$
BY $\langle 2 \rangle 2$ DEF $HistorySummariesMatchRecursion$
$\langle 3 \rangle 3.$ QED
BY $\langle 2 \rangle 4$, $\langle 2 \rangle 5$, $\langle 3 \rangle 1$, $\langle 3 \rangle 2$

Due to language limitations, we do not state or prove the step that ties together the base case and the inductive case into a proof for all cases.

$\langle 1 \rangle 4.$ QED

OMITTED

The *AuthenticatorsMatchUniqueLemma* asserts that there is a unique Memoir-Basic authenticator that matches a particular Memoir-Opt authenticator.

$$
\begin{array}{l}
\textsc{theorem}\ \mathit{AuthenticatorsMatchUniqueLemma} \triangleq \\
\quad \forall\ \mathit{ll1Authenticator1}, \mathit{ll1Authenticator2} \in \mathit{MACType}, \\
\quad\quad \mathit{ll2Authenticator} \in \mathit{MACType}, \\
\quad\quad \mathit{symmetricKey1}, \mathit{symmetricKey2} \in \mathit{SymmetricKeyType}, \\
\quad\quad \mathit{hashBarrier} \in \mathit{HashType} : \\
\quad\quad\quad (\wedge\ \mathit{AuthenticatorsMatch}(\\
\quad\quad\quad\quad\quad \mathit{ll1Authenticator1}, \mathit{ll2Authenticator}, \mathit{symmetricKey1}, \mathit{hashBarrier}) \\
\quad\quad\quad\ \ \wedge\ \mathit{AuthenticatorsMatch}(\\
\quad\quad\quad\quad\quad \mathit{ll1Authenticator2}, \mathit{ll2Authenticator}, \mathit{symmetricKey2}, \mathit{hashBarrier})) \\
\quad\quad\quad \Rightarrow \\
\quad\quad\quad\quad \mathit{ll1Authenticator1} = \mathit{ll1Authenticator2}
\end{array}
$$

To prove the universally quantified expression, we take a set of variables in the appropriate types.

$$
\begin{array}{l}
\langle 1 \rangle 1.\ \textsc{take}\ \mathit{ll1Authenticator1}, \mathit{ll1Authenticator2} \in \mathit{MACType}, \\
\quad\quad\quad \mathit{ll2Authenticator} \in \mathit{MACType}, \\
\quad\quad\quad \mathit{symmetricKey1}, \mathit{symmetricKey2} \in \mathit{SymmetricKeyType}, \\
\quad\quad\quad \mathit{hashBarrier} \in \mathit{HashType}
\end{array}
$$

We assume the antecedent of the implication.

$$
\begin{array}{l}
\langle 1 \rangle 2.\ \textsc{have}\ \wedge\ \mathit{AuthenticatorsMatch}(\\
\quad\quad\quad\quad \mathit{ll1Authenticator1}, \mathit{ll2Authenticator}, \mathit{symmetricKey1}, \mathit{hashBarrier}) \\
\quad\quad\quad \wedge\ \mathit{AuthenticatorsMatch}(\\
\quad\quad\quad\quad \mathit{ll1Authenticator2}, \mathit{ll2Authenticator}, \mathit{symmetricKey2}, \mathit{hashBarrier})
\end{array}
$$

We pick a set of variables that witness the existentials inside the *AuthenticatorsMatch* predicate for authenticator 1.

$$
\begin{array}{l}
\langle 1 \rangle 3.\ \textsc{pick}\ \mathit{stateHash1} \in \mathit{HashType}, \\
\quad\quad\quad \mathit{ll1HistorySummary1} \in \mathit{HashType}, \\
\quad\quad\quad \mathit{ll2HistorySummary1} \in \mathit{HistorySummaryType} : \\
\quad\quad\quad \mathit{AuthenticatorsMatch}(\\
\quad\quad\quad\quad \mathit{ll1Authenticator1}, \mathit{ll2Authenticator}, \mathit{symmetricKey1}, \mathit{hashBarrier})!(\\
\quad\quad\quad\quad \mathit{stateHash1}, \mathit{ll1HistorySummary1}, \mathit{ll2HistorySummary1}) \\
\quad \textsc{by}\ \langle 1 \rangle 2\ \textsc{def}\ \mathit{AuthenticatorsMatch}
\end{array}
$$

We re-state the definitions from the LET in *AuthenticatorsMatch* for authenticator 1.

$$
\begin{array}{l}
\langle 1 \rangle\ \mathit{ll1HistoryStateBinding1} \triangleq \mathit{Hash}(\mathit{ll1HistorySummary1}, \mathit{stateHash1}) \\
\langle 1 \rangle\ \mathit{ll2HistorySummaryHash1} \triangleq \mathit{Hash}(\mathit{ll2HistorySummary1}.\mathit{anchor}, \mathit{ll2HistorySummary1}.\mathit{extension}) \\
\langle 1 \rangle\ \mathit{ll2HistoryStateBinding1} \triangleq \mathit{Hash}(\mathit{ll2HistorySummaryHash1}, \mathit{stateHash1})
\end{array}
$$

We prove the types of the definitions, with the help of the *AuthenticatorsMatchDefsTypeSafeLemma*.

$$
\begin{array}{l}
\langle 1 \rangle 4.\ \wedge\ \mathit{ll1HistoryStateBinding1} \in \mathit{HashType} \\
\quad\quad \wedge\ \mathit{ll2HistorySummaryHash1} \in \mathit{HashType} \\
\quad\quad \wedge\ \mathit{ll2HistoryStateBinding1} \in \mathit{HashType} \\
\quad \textsc{by}\ \langle 1 \rangle 3, \mathit{AuthenticatorsMatchDefsTypeSafeLemma}
\end{array}
$$

We hide the definitions.

$$
\langle 1 \rangle\ \textsc{hide def}\ \mathit{ll1HistoryStateBinding1}, \mathit{ll2HistorySummaryHash1}, \mathit{ll2HistoryStateBinding1}
$$

We pick a set of variables that witness the existentials inside the *AuthenticatorsMatch* predicate for authenticator 2.

$$
\begin{array}{l}
\langle 1 \rangle 5.\ \textsc{pick}\ \mathit{stateHash2} \in \mathit{HashType}, \\
\quad\quad\quad \mathit{ll1HistorySummary2} \in \mathit{HashType}, \\
\quad\quad\quad \mathit{ll2HistorySummary2} \in \mathit{HistorySummaryType} :
\end{array}
$$

$AuthenticatorsMatch($
$ll1Authenticator2$, $ll2Authenticator$, $symmetricKey2$, $hashBarrier$)!(
$stateHash2$, $ll1HistorySummary2$, $ll2HistorySummary2$)
BY $\langle 1 \rangle 2$ DEF $AuthenticatorsMatch$

We re-state the definitions from the LET in $AuthenticatorsMatch$ for authenticator 2.

$\langle 1 \rangle$ $ll1HistoryStateBinding2 \triangleq Hash(ll1HistorySummary2, stateHash2)$
$\langle 1 \rangle$ $ll2HistorySummaryHash2 \triangleq Hash(ll2HistorySummary2.anchor, ll2HistorySummary2.extension)$
$\langle 1 \rangle$ $ll2HistoryStateBinding2 \triangleq Hash(ll2HistorySummaryHash2, stateHash2)$

We prove the types of the definitions, with the help of the $AuthenticatorsMatchDefsTypeSafeLemma$.

$\langle 1 \rangle 6.$ $\land$ $ll1HistoryStateBinding2 \in HashType$
$\land$ $ll2HistorySummaryHash2 \in HashType$
$\land$ $ll2HistoryStateBinding2 \in HashType$
BY $\langle 1 \rangle 3$, $AuthenticatorsMatchDefsTypeSafeLemma$

We hide the definitions.

$\langle 1 \rangle$ HIDE DEF $ll1HistoryStateBinding2$, $ll2HistorySummaryHash2$, $ll2HistoryStateBinding2$

Beginning the proof proper, we first prove that symmetric key 1 and history state binding 1 validate the MAC generated with symmetric key 2 and history state binding 2.

$\langle 1 \rangle 7.$ $ValidateMAC($
$symmetricKey1$,
$ll2HistoryStateBinding1$,
$GenerateMAC(symmetricKey2, ll2HistoryStateBinding2))$
$\langle 2 \rangle 1.$ $ValidateMAC(symmetricKey1, ll2HistoryStateBinding1, ll2Authenticator)$
BY $\langle 1 \rangle 3$ DEF $ll2HistoryStateBinding1$, $ll2HistorySummaryHash1$
$\langle 2 \rangle 2.$ $ll2Authenticator = GenerateMAC(symmetricKey2, ll2HistoryStateBinding2)$
$\langle 3 \rangle 1.$ $ValidateMAC(symmetricKey2, ll2HistoryStateBinding2, ll2Authenticator)$
BY $\langle 1 \rangle 5$ DEF $ll2HistoryStateBinding2$, $ll2HistorySummaryHash2$
$\langle 3 \rangle 2.$ QED
BY $\langle 1 \rangle 1$, $\langle 1 \rangle 6$, $\langle 3 \rangle 1$, $MACConsistent$
$\langle 2 \rangle 3.$ QED
BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$

Next, we prove that the two Memoir-Opt history summary hashes are equal and the two state hashes are equal. This follows from the collision resistance of the MAC functions applied to the previous step, followed by the collision resistance of the hash function.

$\langle 1 \rangle 8.$ $\land$ $ll2HistorySummaryHash1 = ll2HistorySummaryHash2$
$\land$ $stateHash1 = stateHash2$
$\langle 2 \rangle 1.$ $ll2HistoryStateBinding1 = ll2HistoryStateBinding2$
BY $\langle 1 \rangle 1$, $\langle 1 \rangle 4$, $\langle 1 \rangle 6$, $\langle 1 \rangle 7$, $MACCollisionResistant$
$\langle 2 \rangle 2.$ $ll2HistorySummaryHash1 \in HashDomain$
BY $\langle 1 \rangle 4$ DEF $HashDomain$
$\langle 2 \rangle 3.$ $stateHash1 \in HashDomain$
BY $\langle 1 \rangle 3$ DEF $HashDomain$
$\langle 2 \rangle 4.$ $ll2HistorySummaryHash2 \in HashDomain$
BY $\langle 1 \rangle 6$ DEF $HashDomain$
$\langle 2 \rangle 5.$ $stateHash2 \in HashDomain$
BY $\langle 1 \rangle 5$ DEF $HashDomain$
$\langle 2 \rangle 6.$ QED
BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, $\langle 2 \rangle 3$, $\langle 2 \rangle 4$, $\langle 2 \rangle 5$, $HashCollisionResistant$
DEF $ll2HistoryStateBinding1$, $ll2HistoryStateBinding2$

Because the Memoir-Opt history summary hashes are equal, the Memoir-Opt history summaries are equal.

$\langle 1 \rangle 9.$ $ll2HistorySummary1 = ll2HistorySummary2$
$\langle 2 \rangle 1.$ $\land$ $ll2HistorySummary1.anchor = ll2HistorySummary2.anchor$

$\wedge$ *ll2HistorySummary1.extension* = *ll2HistorySummary2.extension*

⟨3⟩1. *ll2HistorySummary1.anchor* ∈ *HashDomain*
BY ⟨1⟩3 DEF *HistorySummaryType*, *HashDomain*

⟨3⟩2. *ll2HistorySummary1.extension* ∈ *HashDomain*
BY ⟨1⟩3 DEF *HistorySummaryType*, *HashDomain*

⟨3⟩3. *ll2HistorySummary2.anchor* ∈ *HashDomain*
BY ⟨1⟩5 DEF *HistorySummaryType*, *HashDomain*

⟨3⟩4. *ll2HistorySummary2.extension* ∈ *HashDomain*
BY ⟨1⟩5 DEF *HistorySummaryType*, *HashDomain*

⟨3⟩5. QED

Ideally, this QED step should just read:

BY ⟨3⟩1, ⟨3⟩2, ⟨3⟩3, ⟨3⟩4, ⟨1⟩8, *HashCollisionResistant* DEF *ll2HistorySummaryHash1*, *ll2HistorySummaryHash2*

However, the prover seems to get a little confused in this instance. We make life easier for the prover by definining some local variables and hiding their definitions before appealing to the *HashCollisionResistant* assumption.

⟨4⟩ $h1a \triangleq$ *ll2HistorySummary1.anchor*
⟨4⟩ $h2a \triangleq$ *ll2HistorySummary1.extension*
⟨4⟩ $h1b \triangleq$ *ll2HistorySummary2.anchor*
⟨4⟩ $h2b \triangleq$ *ll2HistorySummary2.extension*

⟨4⟩1. $h1a$ ∈ *HashDomain*
BY ⟨3⟩1

⟨4⟩2. $h2a$ ∈ *HashDomain*
BY ⟨3⟩2

⟨4⟩3. $h1b$ ∈ *HashDomain*
BY ⟨3⟩3

⟨4⟩4. $h2b$ ∈ *HashDomain*
BY ⟨3⟩4

⟨4⟩5. $Hash(h1a, h2a) = Hash(h1b, h2b)$
BY ⟨1⟩8 DEF *ll2HistorySummaryHash1*, *ll2HistorySummaryHash2*

⟨4⟩6. $h1a = h1b \wedge h2a = h2b$
⟨5⟩ HIDE DEF $h1a$, $h2a$, $h1b$, $h2b$
⟨5⟩1. QED
BY ⟨4⟩1, ⟨4⟩2, ⟨4⟩3, ⟨4⟩4, ⟨4⟩5, *HashCollisionResistant*

⟨4⟩7. QED
BY ⟨4⟩6

⟨2⟩2. QED
BY ⟨1⟩3, ⟨1⟩5, ⟨2⟩1, *HistorySummaryRecordCompositionLemma*

Finally, we prove the equality of the Memoir-Basic state authenticators. This equality follows from the fact that they are generated with identical symmetric keys and identical history state bindings. The equality of the symmetric keys follows from the unforgeability of the *MAC*. The equality of the Memoir-Basic history state bindings follows from the equality of the Memoir-Basic history summaries and the equality of the state hashes. And the equality of the Memoir-Basic history summaries follows from the equality of the Memoir-Opt history summaries, by employing the *HistorySummariesMatchUniqueLemma*.

⟨1⟩10. QED
⟨2⟩1. *ll1HistoryStateBinding1* = *ll1HistoryStateBinding2*
⟨3⟩1. *ll1HistorySummary1* = *ll1HistorySummary2*
⟨4⟩1. *HistorySummariesMatch*(*ll1HistorySummary1*, *ll2HistorySummary1*, *hashBarrier*)
BY ⟨1⟩3

⟨4⟩2. *HistorySummariesMatch*(*ll1HistorySummary2*, *ll2HistorySummary2*, *hashBarrier*)
BY ⟨1⟩5

⟨4⟩3. QED
BY ⟨1⟩3, ⟨1⟩5, ⟨1⟩9, ⟨4⟩1, ⟨4⟩2, *HistorySummariesMatchUniqueLemma*

⟨3⟩2. QED

BY ⟨1⟩8, ⟨3⟩1 DEF *ll1HistoryStateBinding1*, *ll1HistoryStateBinding2*

⟨2⟩2. $symmetricKey1 = symmetricKey2$

BY ⟨1⟩1, ⟨1⟩4, ⟨1⟩6, ⟨1⟩7, *MACUnforgeable*

⟨2⟩3. $ll1Authenticator1 = GenerateMAC(symmetricKey1, ll1HistoryStateBinding1)$

BY ⟨1⟩3 DEF *ll1HistoryStateBinding1*

⟨2⟩4. $ll1Authenticator2 = GenerateMAC(symmetricKey2, ll1HistoryStateBinding2)$

BY ⟨1⟩5 DEF *ll1HistoryStateBinding2*

⟨2⟩5. QED

BY ⟨2⟩1, ⟨2⟩2, ⟨2⟩3, ⟨2⟩4

The *AuthenticatorSetsMatchUniqueLemma* asserts that there is a unique Memoir-Basic set of authenticators that matches a particular Memoir-Opt set of authenticators.

THEOREM *AuthenticatorSetsMatchUniqueLemma* $\triangleq$

$$\begin{array}{l}
\forall\ ll1AuthenticatorSet1, ll1AuthenticatorSet2 \in \text{SUBSET } MACType, \\
\quad ll2AuthenticatorSet \in \text{SUBSET } MACType, \\
\quad symmetricKey \in SymmetricKeyType, \\
\quad hashBarrier \in HashType : \\
\quad\quad (\land\ AuthenticatorSetsMatch(\\
\quad\quad\quad\quad ll1AuthenticatorSet1, ll2AuthenticatorSet, symmetricKey, hashBarrier) \\
\quad\quad\ \ \land\ AuthenticatorSetsMatch(\\
\quad\quad\quad\quad ll1AuthenticatorSet2, ll2AuthenticatorSet, symmetricKey, hashBarrier)) \\
\quad\Rightarrow \\
\quad\quad ll1AuthenticatorSet1 = ll1AuthenticatorSet2
\end{array}$$

To prove the universally quantified expression, we take a set of variables in the appropriate types.

⟨1⟩1. TAKE $ll1AuthenticatorSet1, ll1AuthenticatorSet2 \in \text{SUBSET } MACType,$
$ll2AuthenticatorSet \in \text{SUBSET } MACType,$
$symmetricKey \in SymmetricKeyType,$
$hashBarrier \in HashType$

We assume the antecedent of the implication.

⟨1⟩2. HAVE $\land\ AuthenticatorSetsMatch(ll1AuthenticatorSet1, ll2AuthenticatorSet, symmetricKey, hashBarrier)$
$\land\ AuthenticatorSetsMatch(ll1AuthenticatorSet2, ll2AuthenticatorSet, symmetricKey, hashBarrier)$

The proof has two main steps, which are mirror images of each other. First, we prove that every Memoir-Basic authenticator in set 1 is also in set 2.

⟨1⟩3. $\forall\, ll1Authenticator \in ll1AuthenticatorSet1 :$
$ll1Authenticator \in ll1AuthenticatorSet2$

To prove the universally quantified expression, we take an arbitrary authenticator in Memoir-Basic set 1.

⟨2⟩1. TAKE $ll1Authenticator1 \in ll1AuthenticatorSet1$

Then we pick a matching authenticator in the Memoir-Opt set.

⟨2⟩2. PICK $ll2Authenticator \in ll2AuthenticatorSet :$
$AuthenticatorsMatch(ll1Authenticator1, ll2Authenticator, symmetricKey, hashBarrier)$

BY ⟨1⟩2 DEF *AuthenticatorSetsMatch*

We then pick a matching authenticator in set 2.

⟨2⟩3. PICK $ll1Authenticator2 \in ll1AuthenticatorSet2 :$
$AuthenticatorsMatch(ll1Authenticator2, ll2Authenticator, symmetricKey, hashBarrier)$

BY $\langle 1\rangle 2$ DEF *AuthenticatorSetsMatch*

The two Memoir-Basic authenticators match, by the *AuthenticatorsMatchUniqueLemma*.

$\langle 2\rangle 4.$ $ll1Authenticator1 = ll1Authenticator2$
$\langle 3\rangle 1.$ $ll1Authenticator1 \in MACType$
BY $\langle 1\rangle 1$, $\langle 2\rangle 1$
$\langle 3\rangle 2.$ $ll1Authenticator2 \in MACType$
BY $\langle 1\rangle 1$, $\langle 2\rangle 3$
$\langle 3\rangle 3.$ $ll2Authenticator \in MACType$
BY $\langle 1\rangle 1$, $\langle 2\rangle 2$
$\langle 3\rangle 4.$ QED
BY $\langle 2\rangle 2$, $\langle 2\rangle 3$, $\langle 3\rangle 1$, $\langle 3\rangle 2$, $\langle 3\rangle 3$, *AuthenticatorsMatchUniqueLemma*
$\langle 2\rangle 5.$ QED
BY $\langle 2\rangle 3$, $\langle 2\rangle 4$

Second, we prove that every Memoir-Basic authenticator in set 2 is also in set 1.

$\langle 1\rangle 4.$ $\forall\, ll1Authenticator \in ll1AuthenticatorSet2 :$
$ll1Authenticator \in ll1AuthenticatorSet1$

To prove the universally quantified expression, we take an arbitrary authenticator in Memoir-Basic set 2.

$\langle 2\rangle 1.$ TAKE $ll1Authenticator2 \in ll1AuthenticatorSet2$

Then we pick a matching authenticator in the Memoir-Opt set.

$\langle 2\rangle 2.$ PICK $ll2Authenticator \in ll2AuthenticatorSet :$
$AuthenticatorsMatch($
$ll1Authenticator2, ll2Authenticator, symmetricKey, hashBarrier)$
BY $\langle 1\rangle 2$ DEF *AuthenticatorSetsMatch*

We then pick a matching authenticator in set 1.

$\langle 2\rangle 3.$ PICK $ll1Authenticator1 \in ll1AuthenticatorSet1 :$
$AuthenticatorsMatch($
$ll1Authenticator1, ll2Authenticator, symmetricKey, hashBarrier)$
BY $\langle 1\rangle 2$ DEF *AuthenticatorSetsMatch*

The two Memoir-Basic authenticators match, by the *AuthenticatorsMatchUniqueLemma*.

$\langle 2\rangle 4.$ $ll1Authenticator1 = ll1Authenticator2$
$\langle 3\rangle 1.$ $ll1Authenticator1 \in MACType$
BY $\langle 1\rangle 1$, $\langle 2\rangle 3$
$\langle 3\rangle 2.$ $ll1Authenticator2 \in MACType$
BY $\langle 1\rangle 1$, $\langle 2\rangle 1$
$\langle 3\rangle 3.$ $ll2Authenticator \in MACType$
BY $\langle 1\rangle 1$, $\langle 2\rangle 2$
$\langle 3\rangle 4.$ QED
BY $\langle 2\rangle 2$, $\langle 2\rangle 3$, $\langle 3\rangle 1$, $\langle 3\rangle 2$, $\langle 3\rangle 3$, *AuthenticatorsMatchUniqueLemma*
$\langle 2\rangle 5.$ QED
BY $\langle 2\rangle 3$, $\langle 2\rangle 4$
$\langle 1\rangle 5.$ QED
BY $\langle 1\rangle 3$, $\langle 1\rangle 4$

The *LL2NVRAMLogicalHistorySummaryTypeSafe* lemma asserts that the *LL2NVRAMLogicalHistorySummary* is of *HistorySummaryType* as long as the Memoir-Opt type invariant is satisfied.

THEOREM $LL2NVRAMLogicalHistorySummaryTypeSafe \triangleq$
$\land \quad LL2TypeInvariant \Rightarrow LL2NVRAMLogicalHistorySummary \in HistorySummaryType$
$\land \quad LL2TypeInvariant' \Rightarrow LL2NVRAMLogicalHistorySummary' \in HistorySummaryType$

$\langle 1\rangle 1.$ $LL2TypeInvariant \Rightarrow LL2NVRAMLogicalHistorySummary \in HistorySummaryType$
$\langle 2\rangle 1.$ HAVE $LL2TypeInvariant$
$\langle 2\rangle 2.$ $LL2NVRAM.historySummaryAnchor \in HashType$
BY $\langle 2\rangle 1$, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$
$\langle 2\rangle 3.$ $LL2SPCR \in HashType$
BY $\langle 2\rangle 1$ DEF $LL2TypeInvariant$
$\langle 2\rangle 4.$ $BaseHashValue \in HashType$
BY $ConstantsTypeSafe$
$\langle 2\rangle 5.$ $CrazyHashValue \in HashType$
BY $CrazyHashValueTypeSafe$
$\langle 2\rangle 6.$ QED
BY $\langle 2\rangle 2$, $\langle 2\rangle 3$, $\langle 2\rangle 4$, $\langle 2\rangle 5$ DEF $LL2NVRAMLogicalHistorySummary$, $HistorySummaryType$
$\langle 1\rangle 2.$ $LL2TypeInvariant' \Rightarrow LL2NVRAMLogicalHistorySummary' \in HistorySummaryType$
$\langle 2\rangle 1.$ HAVE $LL2TypeInvariant'$
$\langle 2\rangle 2.$ $LL2NVRAM.historySummaryAnchor' \in HashType$
BY $\langle 2\rangle 1$, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$
$\langle 2\rangle 3.$ $LL2SPCR' \in HashType$
BY $\langle 2\rangle 1$ DEF $LL2TypeInvariant$
$\langle 2\rangle 4.$ $BaseHashValue \in HashType$
BY $ConstantsTypeSafe$
$\langle 2\rangle 5.$ $CrazyHashValue \in HashType$
BY $CrazyHashValueTypeSafe$
$\langle 2\rangle 6.$ QED
BY $\langle 2\rangle 2$, $\langle 2\rangle 3$, $\langle 2\rangle 4$, $\langle 2\rangle 5$ DEF $LL2NVRAMLogicalHistorySummary$, $HistorySummaryType$
$\langle 1\rangle 3.$ QED
BY $\langle 1\rangle 1$, $\langle 1\rangle 2$

The *AuthenticatorInSetLemma* states that if (1) two state authenticators match across the two specs, (2) two authenticator sets match across the two specs, and (3) in the Memoir-Opt spec, the authenticator is an element of the authenticator set, then in the Memoir-Basic spec, the authenticator is an element of the authenticator set.

THEOREM $AuthenticatorInSetLemma \triangleq$
$\forall\ ll1Authenticator \in MACType,$
$ll2Authenticator \in MACType,$
$ll1Authenticators \in \text{SUBSET } MACType,$
$ll2Authenticators \in \text{SUBSET } MACType,$
$symmetricKey1 \in SymmetricKeyType,$
$symmetricKey2 \in SymmetricKeyType,$
$hashBarrier \in HashType :$
$(\land\ AuthenticatorsMatch($
$ll1Authenticator, ll2Authenticator, symmetricKey1, hashBarrier)$
$\land\ AuthenticatorSetsMatch($
$ll1Authenticators, ll2Authenticators, symmetricKey2, hashBarrier)$
$\land\ ll2Authenticator \in ll2Authenticators)$
$\Rightarrow$
$ll1Authenticator \in ll1Authenticators$

To prove the universally quantified expression, we take a set of variables of the appropriate types.

$\langle 1\rangle 1.$ TAKE $ll1Authenticator1 \in MACType,$
$ll2Authenticator1 \in MACType,$
$ll1Authenticators \in \text{SUBSET } MACType,$
$ll2Authenticators \in \text{SUBSET } MACType,$

$symmetricKey1 \in SymmetricKeyType,$
$symmetricKey2 \in SymmetricKeyType,$
$hashBarrier \in HashType$

To prove the implication, we assume the antecedent.

$\langle 1 \rangle 2.$ HAVE $\wedge\ AuthenticatorsMatch($
$ll1Authenticator1,\ ll2Authenticator1,\ symmetricKey1,\ hashBarrier)$
$\wedge\ AuthenticatorSetsMatch($
$ll1Authenticators,\ ll2Authenticators,\ symmetricKey2,\ hashBarrier)$
$\wedge\ ll2Authenticator1 \in ll2Authenticators$

We pick a new Memoir-Basic authenticator that (1) is in the given Memoir-Basic authenticator set and (2) matches the given Memoir-Opt authenticator.

$\langle 1 \rangle 3.$ PICK $ll1Authenticator2 \in ll1Authenticators :$
$AuthenticatorsMatch($
$ll1Authenticator2,\ ll2Authenticator1,\ symmetricKey2,\ hashBarrier)$

We first prove that such an element exists.

$\langle 2 \rangle 1.\ \exists\, ll1Authenticator2 \in ll1Authenticators :$
$AuthenticatorsMatch($
$ll1Authenticator2,\ ll2Authenticator1,\ symmetricKey2,\ hashBarrier)$

By hypothesis of the lemma, the given Memoir-Opt authenticator is an element of the given Memoir-Opt authenticator set.

$\langle 3 \rangle 1.\ ll2Authenticator1 \in ll2Authenticators$
BY $\langle 1 \rangle 2$

Because the two authenticator sets match, there exists a matching Memoir-Basic authenticator for every Memoir-Opt authenticator. This follows from the definition of *AuthenticatorSetsMatch*.

$\langle 3 \rangle 2.\ \forall\, ll2Authenticator2 \in ll2Authenticators :$
$\exists\, ll1Authenticator2 \in ll1Authenticators :$
$AuthenticatorsMatch($
$ll1Authenticator2,\ ll2Authenticator2,\ symmetricKey2,\ hashBarrier)$

$\langle 4 \rangle 1.\ AuthenticatorSetsMatch($
$ll1Authenticators,\ ll2Authenticators,\ symmetricKey2,\ hashBarrier)$
BY $\langle 1 \rangle 2$

$\langle 4 \rangle 2.$ QED
BY $\langle 4 \rangle 1$ DEF $AuthenticatorSetsMatch$

Therefore, there exists a matching Memoir-Basic authenticator for the given Memoir-Opt authenticator.

$\langle 3 \rangle 3.$ QED
BY $\langle 3 \rangle 1,\ \langle 3 \rangle 2$

Because such an element exists, we can pick it.

$\langle 2 \rangle 2.$ QED
BY $\langle 2 \rangle 1$

We prove that the given Memoir-Basic authenticator equals the newly picked Memoir-Basic authenticator.

$\langle 1 \rangle 4.\ ll1Authenticator1 = ll1Authenticator2$

The given Memoir-Basic authenticator matches the given Memoir-Opt authenticator, by hypothesis of the lemma.

$\langle 2 \rangle 1.\ AuthenticatorsMatch($
$ll1Authenticator1,\ ll2Authenticator1,\ symmetricKey1,\ hashBarrier)$
BY $\langle 1 \rangle 2$

The newly picked Memoir-Basic authenticator matches the given Memoir-Opt authenticator, by property of the pick.

$\langle 2 \rangle 2.\ AuthenticatorsMatch($
$ll1Authenticator2,\ ll2Authenticator1,\ symmetricKey2,\ hashBarrier)$
BY $\langle 1 \rangle 3$

Since both Memoir-Basic authenticators match the given Memoir-Opt authenticator, the *AuthenticatorsMatchUniqueLemma* tells us that they must be equal.

$\langle 2 \rangle 3.$ QED

We first have to prove some types for the *AuthenticatorsMatchUniqueLemma*.

$\langle 3 \rangle 1.$ $ll1Authenticator1 \in MACType$
BY $\langle 1 \rangle 1$
$\langle 3 \rangle 2.$ $ll1Authenticator2 \in MACType$
$\langle 4 \rangle 1.$ $ll1Authenticator2 \in ll1Authenticators$
BY $\langle 1 \rangle 3$
$\langle 4 \rangle 2.$ $ll1Authenticators \in$ SUBSET $MACType$
BY $\langle 1 \rangle 1$
$\langle 4 \rangle 3.$ QED
BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$
$\langle 3 \rangle 3.$ $ll2Authenticator1 \in MACType$
BY $\langle 1 \rangle 1$
$\langle 3 \rangle 4.$ $symmetricKey1 \in SymmetricKeyType$
BY $\langle 1 \rangle 1$
$\langle 3 \rangle 5.$ $symmetricKey2 \in SymmetricKeyType$
BY $\langle 1 \rangle 1$
$\langle 3 \rangle 6.$ $hashBarrier \in HashType$
BY $\langle 1 \rangle 1$

Then we can apply the *AuthenticatorsMatchUniqueLemma* directly.

$\langle 3 \rangle 7.$ QED
BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, $\langle 3 \rangle 1$, $\langle 3 \rangle 2$, $\langle 3 \rangle 3$, $\langle 3 \rangle 4$, $\langle 3 \rangle 5$, $\langle 3 \rangle 6$, *AuthenticatorsMatchUniqueLemma*

The newly picked Memoir-Basic authenticator is an element of the given Memoir-Opt authenticator set, by property of the pick.

$\langle 1 \rangle 5.$ $ll1Authenticator2 \in ll1Authenticators$
BY $\langle 1 \rangle 3$

Since the two Memoir-Basic authenticators are equal, and the newly picked authenticator is an element of the given Memoir-Opt authenticator set, it follows that the given authenticator is an element of the given Memoir-Opt authenticator set.

$\langle 1 \rangle 6.$ QED
BY $\langle 1 \rangle 4$, $\langle 1 \rangle 5$

The *AuthenticatorGeneratedLemma* states that if (1) two inputs summaries match across the two specs, (2) two authenticators match across the two specs, and (3) in the Memoir-Opt spec, the authenticator is generated as a *MAC* for the history state binding formed from the history summary and any state hash, then in the Memoir-Basic spec, the authenticator is generated as a *MAC* for the history state binding formed from the history summary and the same state hash.

THEOREM $AuthenticatorGeneratedLemma \triangleq$
$\forall\ stateHash \in HashType,$
$ll1HistorySummary \in HashType,$
$ll2HistorySummary \in HistorySummaryType,$
$ll1Authenticator \in MACType,$
$ll2Authenticator \in MACType,$
$symmetricKey1 \in SymmetricKeyType,$
$symmetricKey2 \in SymmetricKeyType,$
$hashBarrier \in HashType :$

LET
$ll1HistoryStateBinding \triangleq Hash(ll1HistorySummary, stateHash)$
$ll2HistorySummaryHash \triangleq Hash(ll2HistorySummary.anchor, ll2HistorySummary.extension)$
$ll2HistoryStateBinding \triangleq Hash(ll2HistorySummaryHash, stateHash)$
IN
$(\land HistorySummariesMatch(ll1HistorySummary, ll2HistorySummary, hashBarrier)$
$\land AuthenticatorsMatch($
$ll1Authenticator, ll2Authenticator, symmetricKey1, hashBarrier)$
$\land ll2Authenticator = GenerateMAC(symmetricKey2, ll2HistoryStateBinding))$
$\Rightarrow$
$ll1Authenticator = GenerateMAC(symmetricKey2, ll1HistoryStateBinding)$

To prove the universally quantified expression, we take a set of variables of the appropriate types.

$\langle 1 \rangle 1.$ TAKE $stateHash1 \in HashType,$
$ll1HistorySummary1 \in HashType,$
$ll2HistorySummary1 \in HistorySummaryType,$
$ll1Authenticator \in MACType,$
$ll2Authenticator \in MACType,$
$symmetricKey1 \in SymmetricKeyType,$
$symmetricKey2 \in SymmetricKeyType,$
$hashBarrier \in HashType$

We re-state the definitions from the LET in the lemma.

$\langle 1 \rangle$ $ll1HistoryStateBinding1 \triangleq Hash(ll1HistorySummary1, stateHash1)$
$\langle 1 \rangle$ $ll2HistorySummaryHash1 \triangleq Hash(ll2HistorySummary1.anchor, ll2HistorySummary1.extension)$
$\langle 1 \rangle$ $ll2HistoryStateBinding1 \triangleq Hash(ll2HistorySummaryHash1, stateHash1)$

We prove the types of the definitions, with help from the *AuthenticatorsMatchDefsTypeSafeLemma*.

$\langle 1 \rangle 2.$ $\land ll1HistoryStateBinding1 \in HashType$
$\land ll2HistorySummaryHash1 \in HashType$
$\land ll2HistoryStateBinding1 \in HashType$
BY $\langle 1 \rangle 1$, *AuthenticatorsMatchDefsTypeSafeLemma*

We hide the definitions.

$\langle 1 \rangle$ HIDE DEF $ll1HistoryStateBinding1, ll2HistorySummaryHash1, ll2HistoryStateBinding1$

To prove the implication, it suffices to assume the antecedent and prove the consequent.

$\langle 1 \rangle 3.$ SUFFICES
ASSUME
$\land HistorySummariesMatch(ll1HistorySummary1, ll2HistorySummary1, hashBarrier)$
$\land AuthenticatorsMatch($
$ll1Authenticator, ll2Authenticator, symmetricKey1, hashBarrier)$
$\land ll2Authenticator = GenerateMAC(symmetricKey2, ll2HistoryStateBinding1)$
PROVE
$ll1Authenticator = GenerateMAC(symmetricKey2, ll1HistoryStateBinding1)$
BY DEF $ll1HistoryStateBinding1, ll2HistorySummaryHash1, ll2HistoryStateBinding1$

We pick a set of variables that satisfy the extentially quantified variables in *AuthenticatorsMatch*.

$\langle 1 \rangle 4.$ PICK $stateHash2 \in HashType,$
$ll1HistorySummary2 \in HashType,$
$ll2HistorySummary2 \in HistorySummaryType :$
$AuthenticatorsMatch($
$ll1Authenticator, ll2Authenticator, symmetricKey1, hashBarrier)!($
$stateHash2, ll1HistorySummary2, ll2HistorySummary2)!1$

We prove that such a set of variables exists. This follows because *AuthenticatorsMatch* is assumed by the lemma.

$\langle 2 \rangle 1.$ $\exists stateHash2 \in HashType,$

$ll1HistorySummary2 \in HashType$,
$ll2HistorySummary2 \in HistorySummaryType$:
$AuthenticatorsMatch($
$ll1Authenticator, ll2Authenticator, symmetricKey1, hashBarrier)!($
$stateHash2, ll1HistorySummary2, ll2HistorySummary2)!1$

⟨3⟩1. $AuthenticatorsMatch($
$ll1Authenticator, ll2Authenticator, symmetricKey1, hashBarrier)$
BY ⟨1⟩3
⟨3⟩2. QED
BY ⟨3⟩1 DEF $AuthenticatorsMatch$
⟨2⟩2. QED
BY ⟨2⟩1

We re-state the definitions from the LET in $AuthenticatorsMatch$.

⟨1⟩ $ll1HistoryStateBinding2 \triangleq Hash(ll1HistorySummary2, stateHash2)$
⟨1⟩ $ll2HistorySummaryHash2 \triangleq Hash(ll2HistorySummary2.anchor, ll2HistorySummary2.extension)$
⟨1⟩ $ll2HistoryStateBinding2 \triangleq Hash(ll2HistorySummaryHash2, stateHash2)$

We prove the types of the definitions, with help from the $AuthenticatorsMatchDefsTypeSafeLemma$.

⟨1⟩5. $\land\ ll1HistoryStateBinding2 \in HashType$
$\land\ ll2HistorySummaryHash2 \in HashType$
$\land\ ll2HistoryStateBinding2 \in HashType$
BY ⟨1⟩4, $AuthenticatorsMatchDefsTypeSafeLemma$

We hide the definitions.

⟨1⟩ HIDE DEF $ll1HistoryStateBinding2, ll2HistorySummaryHash2, ll2HistoryStateBinding2$

The biggest outer step of the proof is showing that in the Memoir-Opt spec, the given values of history summary and state hash equal the newly picked values of history summary and state hash.

⟨1⟩6. $\land\ ll2HistorySummary1 = ll2HistorySummary2$
$\land\ stateHash1 = stateHash2$

In the Memoir-Opt spec, the given values of history summary hash and state hash equal the newly picked values of history summary hash and state hash.

⟨2⟩1. $\land\ ll2HistorySummaryHash1 = ll2HistorySummaryHash2$
$\land\ stateHash1 = stateHash2$

In the Memoir-Opt spec, the given history state binding equals the newly picked history state binding. This follows from the collision resistance of the MAC.

⟨3⟩1. $ll2HistoryStateBinding1 = ll2HistoryStateBinding2$
⟨4⟩1. $ll2Authenticator = GenerateMAC(symmetricKey2, ll2HistoryStateBinding1)$
BY ⟨1⟩3
⟨4⟩2. $ValidateMAC(symmetricKey1, ll2HistoryStateBinding2, ll2Authenticator)$
BY ⟨1⟩4 DEF $ll2HistoryStateBinding2, ll2HistorySummaryHash2$
⟨4⟩3. QED
⟨5⟩1. $symmetricKey1 \in SymmetricKeyType$
BY ⟨1⟩1
⟨5⟩2. $symmetricKey2 \in SymmetricKeyType$
BY ⟨1⟩1
⟨5⟩3. $ll2HistoryStateBinding1 \in HashType$
BY ⟨1⟩2
⟨5⟩4. $ll2HistoryStateBinding2 \in HashType$
BY ⟨1⟩5
⟨5⟩5. QED
BY ⟨4⟩1, ⟨4⟩2, ⟨5⟩1, ⟨5⟩2, ⟨5⟩3, ⟨5⟩4, $MACCollisionResistant$

The history summary hashes and state hashes are each respectively equal, because the hash is collision resistant.

⟨3⟩2. QED

$\langle 4 \rangle 1.$ $ll2HistorySummaryHash1 \in HashDomain$
$\langle 5 \rangle 1.$ $ll2HistorySummaryHash1 \in HashType$
BY $\langle 1 \rangle 2$
$\langle 5 \rangle 2.$ QED
BY $\langle 5 \rangle 1$ DEF $HashDomain$
$\langle 4 \rangle 2.$ $ll2HistorySummaryHash2 \in HashDomain$
$\langle 5 \rangle 1.$ $ll2HistorySummaryHash2 \in HashType$
BY $\langle 1 \rangle 5$
$\langle 5 \rangle 2.$ QED
BY $\langle 5 \rangle 1$ DEF $HashDomain$
$\langle 4 \rangle 3.$ $stateHash1 \in HashDomain$
$\langle 5 \rangle 1.$ $stateHash1 \in HashType$
BY $\langle 1 \rangle 2$
$\langle 5 \rangle 2.$ QED
BY $\langle 5 \rangle 1$ DEF $HashDomain$
$\langle 4 \rangle 4.$ $stateHash2 \in HashDomain$
$\langle 5 \rangle 1.$ $stateHash2 \in HashType$
BY $\langle 1 \rangle 5$
$\langle 5 \rangle 2.$ QED
BY $\langle 5 \rangle 1$ DEF $HashDomain$
$\langle 4 \rangle 5.$ QED
BY $\langle 3 \rangle 1$, $\langle 4 \rangle 1$, $\langle 4 \rangle 2$, $\langle 4 \rangle 3$, $\langle 4 \rangle 4$, $HashCollisionResistant$
DEF $ll2HistoryStateBinding1$, $ll2HistoryStateBinding2$

In the Memoir-Opt spec, the given value of history summary equals the newly picked value of history summary.

$\langle 2 \rangle 2.$ $ll2HistorySummary1 = ll2HistorySummary2$

The corresponding fields of the two history summaries (the given one and the newly picked one) are equal, because the hash is collision resistant.

$\langle 3 \rangle 1.$ $\land\ ll2HistorySummary1.anchor = ll2HistorySummary2.anchor$
$\land\ ll2HistorySummary1.extension = ll2HistorySummary2.extension$
$\langle 4 \rangle 1.$ $ll2HistorySummaryHash1 = ll2HistorySummaryHash2$
BY $\langle 2 \rangle 1$
$\langle 4 \rangle 2.$ $ll2HistorySummary1.anchor \in HashDomain$
$\langle 5 \rangle 1.$ $ll2HistorySummary1.anchor \in HashType$
BY $\langle 1 \rangle 1$ DEF $HistorySummaryType$
$\langle 5 \rangle 2.$ QED
BY $\langle 5 \rangle 1$ DEF $HashDomain$
$\langle 4 \rangle 3.$ $ll2HistorySummary1.extension \in HashDomain$
$\langle 5 \rangle 1.$ $ll2HistorySummary1.extension \in HashType$
BY $\langle 1 \rangle 1$ DEF $HistorySummaryType$
$\langle 5 \rangle 2.$ QED
BY $\langle 5 \rangle 1$ DEF $HashDomain$
$\langle 4 \rangle 4.$ $ll2HistorySummary2.anchor \in HashDomain$
$\langle 5 \rangle 1.$ $ll2HistorySummary2.anchor \in HashType$
BY $\langle 1 \rangle 4$ DEF $HistorySummaryType$
$\langle 5 \rangle 2.$ QED
BY $\langle 5 \rangle 1$ DEF $HashDomain$
$\langle 4 \rangle 5.$ $ll2HistorySummary2.extension \in HashDomain$
$\langle 5 \rangle 1.$ $ll2HistorySummary2.extension \in HashType$
BY $\langle 1 \rangle 4$ DEF $HistorySummaryType$
$\langle 5 \rangle 2.$ QED
BY $\langle 5 \rangle 1$ DEF $HashDomain$
$\langle 4 \rangle 6.$ QED

Ideally, this QED step should just read:

BY $\langle 4\rangle 1$, $\langle 4\rangle 2$, $\langle 4\rangle 3$, $\langle 4\rangle 4$, $\langle 4\rangle 5$, *HashCollisionResistant* DEF *ll2HistorySummaryHash*1, *ll2HistorySummaryHash*2

However, the prover seems to get a little confused in this instance. We make life easier for the prover by definining some local variables and hiding their definitions before appealing to the *HashCollisionResistant* assumption.

$\langle 5\rangle$ $h1a \triangleq ll2HistorySummary1.anchor$
$\langle 5\rangle$ $h2a \triangleq ll2HistorySummary1.extension$
$\langle 5\rangle$ $h1b \triangleq ll2HistorySummary2.anchor$
$\langle 5\rangle$ $h2b \triangleq ll2HistorySummary2.extension$
$\langle 5\rangle 1.$ $Hash(h1a, h2a) = Hash(h1b, h2b)$
BY $\langle 4\rangle 1$ DEF *ll2HistorySummaryHash*1, *ll2HistorySummaryHash*2
$\langle 5\rangle 2.$ $h1a \in HashDomain$
BY $\langle 4\rangle 2$
$\langle 5\rangle 3.$ $h2a \in HashDomain$
BY $\langle 4\rangle 3$
$\langle 5\rangle 4.$ $h1b \in HashDomain$
BY $\langle 4\rangle 4$
$\langle 5\rangle 5.$ $h2b \in HashDomain$
BY $\langle 4\rangle 5$
$\langle 5\rangle 6.$ $h1a = h1b \land h2a = h2b$
$\langle 6\rangle$ HIDE DEF $h1a$, $h2a$, $h1b$, $h2b$
$\langle 6\rangle 1.$ QED
BY $\langle 5\rangle 1$, $\langle 5\rangle 2$, $\langle 5\rangle 3$, $\langle 5\rangle 4$, $\langle 5\rangle 5$, *HashCollisionResistant*
$\langle 5\rangle 7.$ QED
BY $\langle 5\rangle 6$

Because the fields are equal, the records are equal, but proving this requires that we prove the types of the records and invoke the *HistorySummaryRecordCompositionLemma*.

$\langle 3\rangle 2.$ $ll2HistorySummary1 \in HistorySummaryType$
BY $\langle 1\rangle 1$
$\langle 3\rangle 3.$ $ll2HistorySummary2 \in HistorySummaryType$
BY $\langle 1\rangle 4$
$\langle 3\rangle 4.$ QED
BY $\langle 3\rangle 1$, $\langle 3\rangle 2$, $\langle 3\rangle 3$, *HistorySummaryRecordCompositionLemma*DEF *HistorySummaryType*

For clarity, we restate one of the conjuncts above.

$\langle 2\rangle 3.$ $stateHash1 = stateHash2$
BY $\langle 2\rangle 1$
$\langle 2\rangle 4.$ QED
BY $\langle 2\rangle 2$, $\langle 2\rangle 3$

We then prove that in the Memoir-Basic spec, the given value of history summary equals the newly picked value of history summary. This follows from the *HistorySummariesMatchUniqueLemma*.

$\langle 1\rangle 7.$ $ll1HistorySummary1 = ll1HistorySummary2$
$\langle 2\rangle 1.$ $HistorySummariesMatch(ll1HistorySummary1, ll2HistorySummary1, hashBarrier)$
BY $\langle 1\rangle 3$
$\langle 2\rangle 2.$ $HistorySummariesMatch(ll1HistorySummary2, ll2HistorySummary2, hashBarrier)$
BY $\langle 1\rangle 4$
$\langle 2\rangle 3.$ $ll1HistorySummary1 \in HashType$
BY $\langle 1\rangle 1$
$\langle 2\rangle 4.$ $ll1HistorySummary2 \in HashType$
BY $\langle 1\rangle 4$
$\langle 2\rangle 5.$ $ll2HistorySummary1 \in HistorySummaryType$
BY $\langle 1\rangle 1$

$\langle 2\rangle 6.$ $hashBarrier \in HashType$
BY $\langle 1\rangle 1$
$\langle 2\rangle 7.$ QED
BY $\langle 1\rangle 6$, $\langle 2\rangle 1$, $\langle 2\rangle 2$, $\langle 2\rangle 3$, $\langle 2\rangle 4$, $\langle 2\rangle 5$, $\langle 2\rangle 6$,
$HistorySummariesMatchUniqueLemma$

In the final step, we show in the Memoir-Basic spec that (1) the given value of the history state binding equals the newly picked value of the history state binding, (2) the two symmetric keys are equal, and (3) the given authenticator is generated as a MAC of the newly picked history state binding. This directly implies that the given authenticator is generated as a MAC of the given history state binding, which is the goal of the lemma.

$\langle 1\rangle 8.$ QED
$\langle 2\rangle 1.$ $ll1HistoryStateBinding1 = ll1HistoryStateBinding2$
$\langle 3\rangle 1.$ $ll1HistorySummary1 = ll1HistorySummary2$
BY $\langle 1\rangle 7$
$\langle 3\rangle 2.$ $stateHash1 = stateHash2$
BY $\langle 1\rangle 6$
$\langle 3\rangle 3.$ QED
BY $\langle 3\rangle 1$, $\langle 3\rangle 2$ DEF $ll1HistoryStateBinding1$, $ll1HistoryStateBinding2$
$\langle 2\rangle 2.$ $symmetricKey1 = symmetricKey2$
$\langle 3\rangle 1.$ $ll2Authenticator = GenerateMAC(symmetricKey2, ll2HistoryStateBinding1)$
BY $\langle 1\rangle 3$
$\langle 3\rangle 2.$ $ValidateMAC(symmetricKey1, ll2HistoryStateBinding2, ll2Authenticator)$
BY $\langle 1\rangle 4$ DEF $ll2HistoryStateBinding2$, $ll2HistorySummaryHash2$
$\langle 3\rangle 3.$ QED
$\langle 4\rangle 1.$ $symmetricKey1 \in SymmetricKeyType$
BY $\langle 1\rangle 1$
$\langle 4\rangle 2.$ $symmetricKey2 \in SymmetricKeyType$
BY $\langle 1\rangle 1$
$\langle 4\rangle 3.$ $ll2HistoryStateBinding1 \in HashType$
BY $\langle 1\rangle 2$
$\langle 4\rangle 4.$ $ll2HistoryStateBinding2 \in HashType$
BY $\langle 1\rangle 5$
$\langle 4\rangle 5.$ QED
BY $\langle 3\rangle 1$, $\langle 3\rangle 2$, $\langle 4\rangle 1$, $\langle 4\rangle 2$, $\langle 4\rangle 3$, $\langle 4\rangle 4$, $MACUnforgeable$
$\langle 2\rangle 3.$ $ll1Authenticator = GenerateMAC(symmetricKey1, ll1HistoryStateBinding2)$
BY $\langle 1\rangle 4$, $\langle 1\rangle 6$ DEF $ll1HistoryStateBinding2$
$\langle 2\rangle 4.$ QED
BY $\langle 2\rangle 1$, $\langle 2\rangle 2$, $\langle 2\rangle 3$ DEF $ll1HistoryStateBinding2$

The $AuthenticatorValidatedLemma$ states that if (1) two inputs summaries match across the two specs, (2) two authenticators match across the two specs, and (3) in the Memoir-Opt spec, the authenticator is a valid MAC for the history state binding formed from the history summary and any state hash, then in the Memoir-Basic spec, the authenticator is a valid MAC for the history state binding formed from the history summary and the same state hash.

THEOREM $AuthenticatorValidatedLemma \triangleq$
$\forall$ $stateHash \in HashType$,
$ll1HistorySummary \in HashType$,
$ll2HistorySummary \in HistorySummaryType$,
$ll1Authenticator \in MACType$,
$ll2Authenticator \in MACType$,
$symmetricKey1 \in SymmetricKeyType$,
$symmetricKey2 \in SymmetricKeyType$,

$hashBarrier \in HashType$:

LET

$ll1HistoryStateBinding \triangleq Hash(ll1HistorySummary, stateHash)$

$ll2HistorySummaryHash \triangleq Hash(ll2HistorySummary.anchor, ll2HistorySummary.extension)$

$ll2HistoryStateBinding \triangleq Hash(ll2HistorySummaryHash, stateHash)$

IN

$(\land HistorySummariesMatch(ll1HistorySummary, ll2HistorySummary, hashBarrier)$

$\land AuthenticatorsMatch($

$ll1Authenticator, ll2Authenticator, symmetricKey1, hashBarrier)$

$\land ValidateMAC(symmetricKey2, ll2HistoryStateBinding, ll2Authenticator))$

$\Rightarrow$

$ValidateMAC(symmetricKey2, ll1HistoryStateBinding, ll1Authenticator)$

To prove the universally quantified expression, we take a set of variables of the appropriate types.

$\langle 1 \rangle 1.$ TAKE $stateHash \in HashType$,
$ll1HistorySummary \in HashType$,
$ll2HistorySummary \in HistorySummaryType$,
$ll1Authenticator \in MACType$,
$ll2Authenticator \in MACType$,
$symmetricKey1 \in SymmetricKeyType$,
$symmetricKey2 \in SymmetricKeyType$,
$hashBarrier \in HashType$

We re-state the definitions from the LET in the lemma.

$\langle 1 \rangle$ $ll1HistoryStateBinding \triangleq Hash(ll1HistorySummary, stateHash)$

$\langle 1 \rangle$ $ll2HistorySummaryHash \triangleq Hash(ll2HistorySummary.anchor, ll2HistorySummary.extension)$

$\langle 1 \rangle$ $ll2HistoryStateBinding \triangleq Hash(ll2HistorySummaryHash, stateHash)$

We prove the types of the definitions, with help from the *AuthenticatorsMatchDefsTypeSafeLemma*.

$\langle 1 \rangle 2. \land ll1HistoryStateBinding \in HashType$
$\land ll2HistorySummaryHash \in HashType$
$\land ll2HistoryStateBinding \in HashType$
BY $\langle 1 \rangle 1$, *AuthenticatorsMatchDefsTypeSafeLemma*

We hide the definitions.

$\langle 1 \rangle$ HIDE DEF $ll1HistoryStateBinding, ll2HistorySummaryHash, ll2HistoryStateBinding$

To prove the implication, it suffices to assume the antecedent and prove the consequent.

$\langle 1 \rangle 3.$ SUFFICES

ASSUME

$\land HistorySummariesMatch(ll1HistorySummary, ll2HistorySummary, hashBarrier)$

$\land AuthenticatorsMatch($

$ll1Authenticator, ll2Authenticator, symmetricKey1, hashBarrier)$

$\land ValidateMAC(symmetricKey2, ll2HistoryStateBinding, ll2Authenticator)$

PROVE

$ValidateMAC(symmetricKey2, ll1HistoryStateBinding, ll1Authenticator)$

BY DEF $ll1HistoryStateBinding, ll2HistorySummaryHash, ll2HistoryStateBinding$

Using the *MACConsistent* property, we show that, since the Memoir-Opt authenticator is a valid *MAC* of the Memoir-Opt history state binding, it must have been generated as a *MAC* of this history state binding.

$\langle 1 \rangle 4.$ $ll2Authenticator = GenerateMAC(symmetricKey2, ll2HistoryStateBinding)$

$\langle 2 \rangle 1.$ $ValidateMAC(symmetricKey2, ll2HistoryStateBinding, ll2Authenticator)$
BY $\langle 1 \rangle 3$

$\langle 2 \rangle 2.$ $symmetricKey1 \in SymmetricKeyType$
BY $\langle 1 \rangle 1$

$\langle 2 \rangle 3.$ $symmetricKey2 \in SymmetricKeyType$
BY $\langle 1 \rangle 1$

$\langle 2 \rangle 4.$ $ll2HistoryStateBinding \in HashType$
BY $\langle 1 \rangle 2$
$\langle 2 \rangle 5.$ $ll2Authenticator \in MACType$
BY $\langle 1 \rangle 1$
$\langle 2 \rangle 6.$ QED
BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, $\langle 2 \rangle 3$, $\langle 2 \rangle 4$, $\langle 2 \rangle 5$, *MACConsistent*

Then, using the *AuthenticatorGeneratedLemma*, we show that the Memoir-Basic authenticator is generated as a *MAC* of the Memoir-Basic history state binding.

$\langle 1 \rangle 5.$ $ll1Authenticator = GenerateMAC(symmetricKey2, ll1HistoryStateBinding)$
$\langle 2 \rangle 1.$ $HistorySummariesMatch(ll1HistorySummary, ll2HistorySummary, hashBarrier)$
BY $\langle 1 \rangle 3$
$\langle 2 \rangle 2.$ $AuthenticatorsMatch($
$ll1Authenticator, ll2Authenticator, symmetricKey1, hashBarrier)$
BY $\langle 1 \rangle 3$
$\langle 2 \rangle 3.$ QED

Ideally, this QED step should just read:

BY $\langle 1 \rangle 1$, $\langle 1 \rangle 4$, $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, *AuthenticatorGeneratedLemma* DEF *ll1HistoryStateBinding*, *ll2HistorySummaryHash*, *ll2HistoryStateBinding*

However, the prover seems to get a little confused in this instance. We make life easier for the prover by explicitly staging the instantiation of the quantified variables within the definition of *AuthenticatorGeneratedLemma*.

$\langle 3 \rangle 1.$ $\forall$ $samLL2Authenticator \in MACType,$
$samSymmetricKey2 \in SymmetricKeyType,$
$samHashBarrier \in HashType :$
$AuthenticatorGeneratedLemma!(stateHash, ll1HistorySummary,$
$ll2HistorySummary, ll1Authenticator, samLL2Authenticator,$
$symmetricKey1, samSymmetricKey2, samHashBarrier)!1$
BY $\langle 1 \rangle 1$, *AuthenticatorGeneratedLemma*
$\langle 3 \rangle 2.$ $AuthenticatorGeneratedLemma!(stateHash, ll1HistorySummary,$
$ll2HistorySummary, ll1Authenticator, ll2Authenticator,$
$symmetricKey1, symmetricKey2, hashBarrier)!1$
BY $\langle 1 \rangle 1$, $\langle 3 \rangle 1$
$\langle 3 \rangle 3.$ QED
BY $\langle 1 \rangle 1$, $\langle 1 \rangle 4$, $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, $\langle 3 \rangle 2$
DEF *ll1HistoryStateBinding*, *ll2HistorySummaryHash*, *ll2HistoryStateBinding*

Finally, using the *MACComplete* property, we show that the Memoir-Basic authenticator is a valid *MAC* of the Memoir-Basic history state binding.

$\langle 1 \rangle 6.$ QED
$\langle 2 \rangle 1.$ $symmetricKey2 \in SymmetricKeyType$
BY $\langle 1 \rangle 1$
$\langle 2 \rangle 2.$ $ll1HistoryStateBinding \in HashType$
BY $\langle 1 \rangle 2$
$\langle 2 \rangle 3.$ QED
BY $\langle 1 \rangle 5$, $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, *MACComplete*DEF *ll1HistoryStateBinding*

The *HistorySummariesMatchAcrossCheckpointLemma* asserts that for any pair of Memoir-Opt history summaries for which one is a checkpoint of the other, there is a unique Memoir-Basic history summary that matches either Memoir-Opt history summary.

THEOREM *HistorySummariesMatchAcrossCheckpointLemma* $\triangleq$

$\forall$ $ll1HistorySummary1$, $ll1HistorySummary2 \in HashType$,
$ll2HistorySummary1$, $ll2HistorySummary2 \in HistorySummaryType$,
$hashBarrier \in HashType$:
($\wedge$ $HistorySummariesMatch(ll1HistorySummary1, ll2HistorySummary1, hashBarrier)$
$\wedge$ $HistorySummariesMatch(ll1HistorySummary2, ll2HistorySummary2, hashBarrier)$
$\wedge$ $ll2HistorySummary2 = Checkpoint(ll2HistorySummary1))$
$\Rightarrow$
$ll1HistorySummary1 = ll1HistorySummary2$

To prove the universally quantified expression, we take a set of variables of the appropriate types.

$\langle 1 \rangle 1.$ TAKE $ll1HistorySummary1$, $ll1HistorySummary2 \in HashType$,
$ll2HistorySummary1$, $ll2HistorySummary2 \in HistorySummaryType$,
$hashBarrier \in HashType$

We assume the antecedent.

$\langle 1 \rangle 2.$ HAVE $\wedge$ $HistorySummariesMatch(ll1HistorySummary1, ll2HistorySummary1, hashBarrier)$
$\wedge$ $HistorySummariesMatch(ll1HistorySummary2, ll2HistorySummary2, hashBarrier)$
$\wedge$ $ll2HistorySummary2 = Checkpoint(ll2HistorySummary1)$

There are two separate cases, one for the base case of the *HistorySummariesMatch* predicate and one for the recursion. The base case is trivial.

$\langle 1 \rangle$ $ll2InitialHistorySummary \triangleq [anchor \mapsto BaseHashValue, extension \mapsto BaseHashValue]$
$\langle 1 \rangle 3.$ CASE $ll2HistorySummary1 = ll2InitialHistorySummary$
$\langle 2 \rangle 1.$ $ll2HistorySummary2 = Checkpoint(ll2HistorySummary1)$
BY $\langle 1 \rangle 2$
$\langle 2 \rangle 2.$ $ll2HistorySummary1.extension = BaseHashValue$
BY $\langle 1 \rangle 3$
$\langle 2 \rangle 3.$ $ll2HistorySummary1 = ll2HistorySummary2$
BY $\langle 2 \rangle 1$, $\langle 2 \rangle 2$ DEF *Checkpoint*
$\langle 2 \rangle 4.$ QED
BY $\langle 1 \rangle 1$, $\langle 1 \rangle 2$, $\langle 2 \rangle 3$, *HistorySummariesMatchUniqueLemma*

The recursive case is fairly involved.

$\langle 1 \rangle 4.$ CASE $ll2HistorySummary1 \neq ll2InitialHistorySummary$

We'll first pick values for the existential variables inside the *HistorySummariesMatchRecursion* predicate that satisfy the predicate for the 1 variables in the antecedent of the theorem. We know the *HistorySummariesMatchRecursion* predicate holds, because the Memoir-Opt history summary does not equal the initial history summary, according to this case.

$\langle 2 \rangle 1.$ PICK $input1 \in InputType$,
$ll1PreviousHistorySummary1 \in HashType$,
$ll2PreviousHistorySummary1 \in HistorySummaryType$:
$HistorySummariesMatchRecursion($
$ll1HistorySummary1, ll2HistorySummary1, hashBarrier)!($
$input1, ll1PreviousHistorySummary1, ll2PreviousHistorySummary1)$
$\langle 3 \rangle 1.$ $\wedge$ $ll1HistorySummary1 \in HashType$
$\wedge$ $ll2HistorySummary1 \in HistorySummaryType$
$\wedge$ $hashBarrier \in HashType$
BY $\langle 1 \rangle 1$
$\langle 3 \rangle 2.$ $HistorySummariesMatch(ll1HistorySummary1, ll2HistorySummary1, hashBarrier)$
BY $\langle 1 \rangle 2$
$\langle 3 \rangle 3.$ $ll2HistorySummary1 \neq ll2InitialHistorySummary$
BY $\langle 1 \rangle 4$
$\langle 3 \rangle 4.$ QED
BY $\langle 3 \rangle 1$, $\langle 3 \rangle 2$, $\langle 3 \rangle 3$, *HistorySummariesMatchDefinition*
DEF *HistorySummariesMatchRecursion*

We'll next pick values for the existential variables inside the *HistorySummariesMatchRecursion* predicate that satisfy the predicate for the 2 variables in the antecedent of the theorem. We know the *HistorySummariesMatchRecursion* predicate holds, because the Memoir-Opt history summary does not equal the initial history summary, but proving this latter point is slightly involved.

$\langle 2\rangle 2.$ PICK $input2 \in InputType,$
$\quad ll1PreviousHistorySummary2 \in HashType,$
$\quad ll2PreviousHistorySummary2 \in HistorySummaryType :$
$\quad HistorySummariesMatchRecursion($
$\quad\quad ll1HistorySummary2, ll2HistorySummary2, hashBarrier)!($
$\quad\quad input2, ll1PreviousHistorySummary2, ll2PreviousHistorySummary2)$

$\langle 3\rangle 1. \land\ ll1HistorySummary2 \in HashType$
$\quad \land\ ll2HistorySummary2 \in HistorySummaryType$
$\quad \land\ hashBarrier \in HashType$
BY $\langle 1\rangle 1$

$\langle 3\rangle 2.$ $HistorySummariesMatch(ll1HistorySummary2, ll2HistorySummary2, hashBarrier)$
BY $\langle 1\rangle 2$

$\langle 3\rangle 3.$ $ll2HistorySummary2 \neq ll2InitialHistorySummary$

Inputs summary 2 does not equal the initial history summary because its anchor field does not equal the base hash value.

$\langle 4\rangle 1.$ $ll2HistorySummary2.anchor \neq BaseHashValue$

We use two more levels of case analysis. For the first level, we show that at least one of the two fields in history summary 1 is not equal to the base hash value.

$\langle 5\rangle 1. \lor\ ll2HistorySummary1.anchor \neq BaseHashValue$
$\quad \lor\ ll2HistorySummary1.extension \neq BaseHashValue$

$\langle 6\rangle 1.$ $ll2HistorySummary1 \in HistorySummaryType$
BY $\langle 3\rangle 1$

$\langle 6\rangle 2.$ QED
BY $\langle 1\rangle 4$, $\langle 6\rangle 1$, *HistorySummaryRecordCompositionLemma*DEF *HistorySummaryType*

We consider the case in which the extension field of history summary 1 is not equal to the base hash value. In this case, the anchor field of history summary 2 is formed by the hash function, so it cannot be equal to the base hash value.

$\langle 5\rangle 2.$ CASE $ll2HistorySummary1.extension \neq BaseHashValue$

$\langle 6\rangle 1.$ $ll2HistorySummary2.anchor =$
$\quad Hash(ll2HistorySummary1.anchor, ll2HistorySummary1.extension)$

$\langle 7\rangle 1.$ $ll2HistorySummary2 = Checkpoint(ll2HistorySummary1)$
BY $\langle 1\rangle 2$

$\langle 7\rangle 2.$ QED
BY $\langle 5\rangle 2$, $\langle 7\rangle 1$ DEF *Checkpoint*

$\langle 6\rangle 2.$ $ll2HistorySummary1.anchor \in HashDomain$
BY $\langle 1\rangle 1$ DEF *HistorySummaryType*, *HashDomain*

$\langle 6\rangle 3.$ $ll2HistorySummary1.extension \in HashDomain$
BY $\langle 1\rangle 1$ DEF *HistorySummaryType*, *HashDomain*

$\langle 6\rangle 4.$ QED
BY $\langle 6\rangle 1$, $\langle 6\rangle 2$, $\langle 6\rangle 3$, *BaseHashValueUnique*

We consider the case in which the anchor field of history summary 1 is not equal to the base hash value. We consider two sub-cases.

$\langle 5\rangle 3.$ CASE $ll2HistorySummary1.anchor \neq BaseHashValue$

In the first sub-case, the extension field of history summary 1 is also not equal to the base hash value. We already proved this case above.

$\langle 6\rangle 1.$ CASE $ll2HistorySummary1.extension \neq BaseHashValue$
BY $\langle 5\rangle 2$, $\langle 6\rangle 1$

In the second sub-case, the extension field of history summary 1 equals the base hash value, so the *Checkpoint* operator acts as an identity operator. Thus, the anchor field of history summary 2 equals the anchor field of history summmary 1, which is not equal to the base hash value.

$\langle 6\rangle 2.$ CASE $ll2HistorySummary1.extension = BaseHashValue$
$\langle 7\rangle 1.$ $ll2HistorySummary2.anchor = ll2HistorySummary1.anchor$
$\langle 8\rangle 1.$ $ll2HistorySummary2 = Checkpoint(ll2HistorySummary1)$
BY $\langle 1\rangle 2$
$\langle 8\rangle 2.$ QED
BY $\langle 6\rangle 2$, $\langle 8\rangle 1$ DEF *Checkpoint*
$\langle 7\rangle 2.$ QED
BY $\langle 5\rangle 3$, $\langle 7\rangle 1$
$\langle 6\rangle 3.$ QED
BY $\langle 6\rangle 1$, $\langle 6\rangle 2$
$\langle 5\rangle 4.$ QED
BY $\langle 5\rangle 1$, $\langle 5\rangle 2$, $\langle 5\rangle 3$
$\langle 4\rangle 2.$ $ll2InitialHistorySummary.anchor = BaseHashValue$
OBVIOUS
$\langle 4\rangle 3.$ QED
BY $\langle 4\rangle 1$, $\langle 4\rangle 2$
$\langle 3\rangle 4.$ QED
BY $\langle 3\rangle 1$, $\langle 3\rangle 2$, $\langle 3\rangle 3$, *HistorySummariesMatchDefinition*
DEF *HistorySummariesMatchRecursion*

We prove that the Memoir-Opt previous history summaries are equal, and the inputs are equal.

$\langle 2\rangle 3.$ $\land\ ll2PreviousHistorySummary1 = ll2PreviousHistorySummary2$
$\land\ input1 = input2$

Several useful facts follow directly from the theorem's assumptions and the above picks.

$\langle 3\rangle 1.$ $LL2HistorySummaryIsSuccessor($
$ll2HistorySummary1, ll2PreviousHistorySummary1, input1, hashBarrier)$
BY $\langle 2\rangle 1$
$\langle 3\rangle 2.$ $LL2HistorySummaryIsSuccessor($
$ll2HistorySummary2, ll2PreviousHistorySummary2, input2, hashBarrier)$
BY $\langle 2\rangle 2$
$\langle 3\rangle 3.$ $ll2HistorySummary2 = Checkpoint(ll2HistorySummary1)$
BY $\langle 1\rangle 2$

We re-state the definitions from *LL2HistorySummaryIsSuccessor* for history summary 1.

$\langle 3\rangle$ $ll2SuccessorHistorySummary1 \triangleq Successor(ll2PreviousHistorySummary1, input1, hashBarrier)$
$\langle 3\rangle$ $ll2CheckpointedSuccessorHistorySummary1 \triangleq Checkpoint(ll2SuccessorHistorySummary1)$

We hide the definitions.

$\langle 3\rangle$ HIDE DEF $ll2SuccessorHistorySummary1$, $ll2CheckpointedSuccessorHistorySummary1$

We re-state the definitions from *Successor* for history summary 1.

$\langle 3\rangle$ $securedInput1 \triangleq Hash(hashBarrier, input1)$
$\langle 3\rangle$ $newAnchor1 \triangleq ll2PreviousHistorySummary1.anchor$
$\langle 3\rangle$ $newExtension1 \triangleq Hash(ll2PreviousHistorySummary1.extension, securedInput1)$
$\langle 3\rangle$ $ll2NewHistorySummary1 \triangleq [$
$anchor \mapsto newAnchor1,$
$extension \mapsto newExtension1]$

We prove the types of the definitions from *Successor* and the definition of *ll2SuccessorHistorySummary1*, with help from the *SuccessorDefsTypeSafeLemma*.

$\langle 3\rangle 4.$ $\land\ securedInput1 \in HashType$
$\land\ newAnchor1 \in HashType$
$\land\ newExtension1 \in HashType$

$\wedge\ \ ll2NewHistorySummary1 \in HistorySummaryType$
$\wedge\ \ ll2NewHistorySummary1.anchor \in HashType$
$\wedge\ \ ll2NewHistorySummary1.extension \in HashType$
BY $\langle 1\rangle 1$, $\langle 2\rangle 1$, *SuccessorDefsTypeSafeLemma*

$\langle 3\rangle 5.$ $ll2SuccessorHistorySummary1 \in HistorySummaryType$
BY $\langle 3\rangle 4$ DEF *ll2SuccessorHistorySummary1*, *Successor*

We hide the definitions.

$\langle 3\rangle$ HIDE DEF *securedInput1*, *newAnchor1*, *newExtension1*, *ll2NewHistorySummary1*

We re-state the definitions from *Checkpoint* for history summary 1.

$\langle 3\rangle$ $checkpointedAnchor1 \triangleq$
$Hash(ll2SuccessorHistorySummary1.anchor,\ ll2SuccessorHistorySummary1.extension)$

$\langle 3\rangle$ $ll2CheckpointedHistorySummary1 \triangleq [$
$anchor \mapsto checkpointedAnchor1,$
$extension \mapsto BaseHashValue]$

We prove the types of the definitions from *Checkpoint* and the definition of *ll2CheckpointedSuccessorHistorySummary1*, with help from the *CheckpointDefsTypeSafeLemma*.

$\langle 3\rangle 6.$ $\wedge\ \ checkpointedAnchor1 \in HashType$
$\wedge\ \ ll2CheckpointedHistorySummary1 \in HistorySummaryType$
$\wedge\ \ ll2CheckpointedHistorySummary1.anchor \in HashType$
$\wedge\ \ ll2CheckpointedHistorySummary1.extension \in HashType$
BY $\langle 3\rangle 5$, *CheckpointDefsTypeSafeLemma*

$\langle 3\rangle 7.$ $ll2CheckpointedSuccessorHistorySummary1 \in HistorySummaryType$
BY $\langle 3\rangle 5$, $\langle 3\rangle 6$ DEF *ll2CheckpointedSuccessorHistorySummary1*, *Checkpoint*

We hide the definitions.

$\langle 3\rangle$ HIDE DEF *checkpointedAnchor1*, *ll2CheckpointedHistorySummary1*

We re-state the definitions from *LL2HistorySummaryIsSuccessor* for history summary 2.

$\langle 3\rangle$ $ll2SuccessorHistorySummary2 \triangleq Successor(ll2PreviousHistorySummary2,\ input2,\ hashBarrier)$
$\langle 3\rangle$ $ll2CheckpointedSuccessorHistorySummary2 \triangleq Checkpoint(ll2SuccessorHistorySummary2)$

We hide the definitions.

$\langle 3\rangle$ HIDE DEF *ll2SuccessorHistorySummary2*, *ll2CheckpointedSuccessorHistorySummary2*

We re-state the definitions from *Successor* for history summary 2.

$\langle 3\rangle$ $securedInput2 \triangleq Hash(hashBarrier,\ input2)$
$\langle 3\rangle$ $newAnchor2 \triangleq ll2PreviousHistorySummary2.anchor$
$\langle 3\rangle$ $newExtension2 \triangleq Hash(ll2PreviousHistorySummary2.extension,\ securedInput2)$
$\langle 3\rangle$ $ll2NewHistorySummary2 \triangleq [$
$anchor \mapsto newAnchor2,$
$extension \mapsto newExtension2]$

We prove the types of the definitions from *Successor* and the definition of *ll2SuccessorHistorySummary2*, with help from the *SuccessorDefsTypeSafeLemma*.

$\langle 3\rangle 8.$ $\wedge\ \ securedInput2 \in HashType$
$\wedge\ \ newAnchor2 \in HashType$
$\wedge\ \ newExtension2 \in HashType$
$\wedge\ \ ll2NewHistorySummary2 \in HistorySummaryType$
$\wedge\ \ ll2NewHistorySummary2.anchor \in HashType$
$\wedge\ \ ll2NewHistorySummary2.extension \in HashType$
BY $\langle 1\rangle 1$, $\langle 2\rangle 1$, *SuccessorDefsTypeSafeLemma*

$\langle 3\rangle 9.$ $ll2SuccessorHistorySummary2 \in HistorySummaryType$
BY $\langle 3\rangle 8$ DEF *ll2SuccessorHistorySummary2*, *Successor*

We hide the definitions.

$\langle 3\rangle$ HIDE DEF *securedInput2*, *newAnchor2*, *newExtension2*, *ll2NewHistorySummary2*

We re-state the definitions from *Checkpoint* for history summary 2.

$\langle 3 \rangle$ $checkpointedAnchor2 \triangleq$
$\quad Hash(ll2SuccessorHistorySummary2.anchor,\ ll2SuccessorHistorySummary2.extension)$

$\langle 3 \rangle$ $ll2CheckpointedHistorySummary2 \triangleq [$
$\quad anchor \mapsto checkpointedAnchor2,$
$\quad extension \mapsto BaseHashValue]$

We prove the types of the definitions from *Checkpoint* and the definition of *ll2CheckpointedSuccessorHistorySummary2*, with help from the *CheckpointDefsTypeSafeLemma*.

$\langle 3 \rangle 10.$ $\land\ checkpointedAnchor2 \in HashType$
$\quad \land\ ll2CheckpointedHistorySummary2 \in HistorySummaryType$
$\quad \land\ ll2CheckpointedHistorySummary2.anchor \in HashType$
$\quad \land\ ll2CheckpointedHistorySummary2.extension \in HashType$
$\quad$ BY $\langle 3 \rangle 9$, *CheckpointDefsTypeSafeLemma*

$\langle 3 \rangle 11.$ $ll2CheckpointedSuccessorHistorySummary2 \in HistorySummaryType$
$\quad$ BY $\langle 3 \rangle 9$, $\langle 3 \rangle 10$ DEF *ll2CheckpointedSuccessorHistorySummary2*, *Checkpoint*

We hide the definitions.

$\langle 3 \rangle$ HIDE DEF *checkpointedAnchor2*, *ll2CheckpointedHistorySummary2*

The most critical part of the proof is showing that the checkpointed successor history summaries, as defined in the LET within the *LL2HistorySummaryIsSuccessor* predicate, are equal. This requires separately considering the cases of whether the extension field of *ll2HistorySummary* equals the base hash value.

$\langle 3 \rangle 12.$ $ll2CheckpointedSuccessorHistorySummary1 = ll2CheckpointedSuccessorHistorySummary2$

In the first case, the extension field of *ll2HistorySummary* equals the base hash value.

$\langle 4 \rangle 1.$ CASE $ll2HistorySummary1.extension = BaseHashValue$

The *LL2HistorySummaryIsSuccessor* predicate expresses a disjunction. An history summary can be a successor either by being a direct successor or by being a checkpoint of a successor. We prove that, in this case, history summary 1 is a checkpoint of a successor. It cannot be a direct successor because its extension field is the base hash value, and any direct successor will have a non-base extension field.

$\langle 5 \rangle 1.$ $ll2HistorySummary1 = ll2CheckpointedSuccessorHistorySummary1$
$\quad \langle 6 \rangle 1.$ $\lor\ ll2HistorySummary1 = ll2SuccessorHistorySummary1$
$\quad\quad \lor\ ll2HistorySummary1 = ll2CheckpointedSuccessorHistorySummary1$
$\quad\quad$ BY $\langle 3 \rangle 1$
$\quad\quad$ DEF *LL2HistorySummaryIsSuccessor*, *ll2SuccessorHistorySummary1*, *ll2CheckpointedSuccessorHistorySummary1*
$\quad \langle 6 \rangle 2.$ $ll2HistorySummary1 \neq ll2SuccessorHistorySummary1$
$\quad\quad \langle 7 \rangle 1.$ $ll2SuccessorHistorySummary1.extension \neq BaseHashValue$
$\quad\quad\quad$ BY $\langle 1 \rangle 1$, $\langle 2 \rangle 1$, *SuccessorHasNonBaseExtensionLemma*DEF *ll2SuccessorHistorySummary1*
$\quad\quad \langle 7 \rangle 2.$ QED
$\quad\quad\quad$ BY $\langle 4 \rangle 1$, $\langle 7 \rangle 1$
$\quad \langle 6 \rangle 3.$ QED
$\quad\quad$ BY $\langle 6 \rangle 1$, $\langle 6 \rangle 2$

Inputs summary 2 is also a checkpoint of a successor, by the same argument as above for history summary 1. The one twist is that we need to prove that the its extension field is the base hash value, which follows from the fact that - when history summary 1 has a base extension field, history summaries 1 and 2 are equal.

$\langle 5 \rangle 2.$ $ll2HistorySummary2 = ll2CheckpointedSuccessorHistorySummary2$
$\quad \langle 6 \rangle 1.$ $\lor\ ll2HistorySummary2 = ll2SuccessorHistorySummary2$
$\quad\quad \lor\ ll2HistorySummary2 = ll2CheckpointedSuccessorHistorySummary2$
$\quad\quad$ BY $\langle 3 \rangle 2$
$\quad\quad$ DEF *LL2HistorySummaryIsSuccessor*, *ll2SuccessorHistorySummary2*, *ll2CheckpointedSuccessorHistorySummary2*
$\quad \langle 6 \rangle 2.$ $ll2HistorySummary2 \neq ll2SuccessorHistorySummary2$
$\quad\quad \langle 7 \rangle 1.$ $ll2SuccessorHistorySummary2.extension \neq BaseHashValue$

BY $\langle 1 \rangle 1$, $\langle 2 \rangle 2$, *SuccessorHasNonBaseExtensionLemma* DEF *ll2SuccessorHistorySummary2*

$\langle 7 \rangle 2$. QED

$\langle 8 \rangle 1$. $ll2HistorySummary2.extension = BaseHashValue$

$\langle 9 \rangle 1$. $ll2HistorySummary1 = ll2HistorySummary2$

BY $\langle 3 \rangle 3$, $\langle 4 \rangle 1$ DEF *Checkpoint*

$\langle 9 \rangle 2$. QED

BY $\langle 4 \rangle 1$, $\langle 9 \rangle 1$

$\langle 8 \rangle 2$. QED

BY $\langle 7 \rangle 1$, $\langle 8 \rangle 1$

$\langle 6 \rangle 3$. QED

BY $\langle 6 \rangle 1$, $\langle 6 \rangle 2$

The conclusion follows from the above-proven equalities, given the fact that, when history summary 1 has a base extension field, the *Checkpoint* operator acts as an identity operator, so history summaries 1 and 2 are equal.

$\langle 5 \rangle 3$. QED

$\langle 6 \rangle 1$. $ll2HistorySummary1 = ll2HistorySummary2$

BY $\langle 3 \rangle 3$, $\langle 4 \rangle 1$ DEF *Checkpoint*

$\langle 6 \rangle 2$. QED

BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$, $\langle 6 \rangle 1$

In the second case, the extension field of *ll2HistorySummary* does not equal the base hash value.

$\langle 4 \rangle 2$. CASE $ll2HistorySummary1.extension \neq BaseHashValue$

The *LL2HistorySummaryIsSuccessor* predicate expresses a disjunction. An history summary can be a successor either by being a direct successor or by being a checkpoint of a successor. We prove that, in this case, history summary 1 is a direct successor. It cannot be a checkpoint because its extension field is not the base hash value, and any checkpoint will have a base extension field.

$\langle 5 \rangle 1$. $ll2HistorySummary1 = ll2SuccessorHistorySummary1$

$\langle 6 \rangle 1$. $\lor\ ll2HistorySummary1 = ll2SuccessorHistorySummary1$
$\quad\lor\ ll2HistorySummary1 = ll2CheckpointedSuccessorHistorySummary1$

BY $\langle 3 \rangle 1$

DEF *LL2HistorySummaryIsSuccessor*, *ll2SuccessorHistorySummary1*, *ll2CheckpointedSuccessorHistorySummary1*

$\langle 6 \rangle 2$. $ll2HistorySummary1 \neq ll2CheckpointedSuccessorHistorySummary1$

$\langle 7 \rangle 1$. $ll2CheckpointedSuccessorHistorySummary1.extension = BaseHashValue$

BY $\langle 3 \rangle 5$, *CheckpointHasBaseExtensionLemma*

DEF *ll2CheckpointedSuccessorHistorySummary1*

$\langle 7 \rangle 2$. QED

BY $\langle 4 \rangle 2$, $\langle 7 \rangle 1$

$\langle 6 \rangle 3$. QED

BY $\langle 6 \rangle 1$, $\langle 6 \rangle 2$

Inputs summary 2 is a checkpoint of a successor, by the reverse of the above argument for history summary 1. We know that the extension field of history summary 2 must be the base hash value, because it is a checkpoint of history summary 1.

$\langle 5 \rangle 2$. $ll2HistorySummary2 = ll2CheckpointedSuccessorHistorySummary2$

$\langle 6 \rangle 1$. $\lor\ ll2HistorySummary2 = ll2SuccessorHistorySummary2$
$\quad\lor\ ll2HistorySummary2 = ll2CheckpointedSuccessorHistorySummary2$

BY $\langle 3 \rangle 2$

DEF *LL2HistorySummaryIsSuccessor*, *ll2SuccessorHistorySummary2*, *ll2CheckpointedSuccessorHistorySummary2*

$\langle 6 \rangle 2$. $ll2HistorySummary2 \neq ll2SuccessorHistorySummary2$

$\langle 7 \rangle 1$. $ll2SuccessorHistorySummary2.extension \neq BaseHashValue$

BY $\langle 1 \rangle 1$, $\langle 2 \rangle 2$, *SuccessorHasNonBaseExtensionLemma* DEF *ll2SuccessorHistorySummary2*

$\langle 7 \rangle 2$. $ll2HistorySummary2.extension = BaseHashValue$

BY ⟨1⟩1, ⟨1⟩2, *CheckpointHasBaseExtensionLemma*

⟨7⟩3. QED

BY ⟨7⟩1, ⟨7⟩2

⟨6⟩3. QED

BY ⟨6⟩1, ⟨6⟩2

The conclusion follows from the above-proven equalities, given the parallel construction of checkpoint from history summary in both the history summaries in the antecedent of the theorem and the history summaries defined within the LET of the *LL2HistorySummaryIsSuccessor* predicate.

⟨5⟩3. QED

⟨6⟩1. $ll2CheckpointedSuccessorHistorySummary1 = Checkpoint(ll2HistorySummary1)$

BY ⟨5⟩1 DEF *ll2CheckpointedSuccessorHistorySummary1*

⟨6⟩2. $ll2HistorySummary2 = Checkpoint(ll2HistorySummary1)$

BY ⟨1⟩2

⟨6⟩3. $ll2CheckpointedSuccessorHistorySummary2 = ll2HistorySummary2$

BY ⟨5⟩2

⟨6⟩4. QED

BY ⟨6⟩1, ⟨6⟩2, ⟨6⟩3

The two cases are exhaustive.

⟨4⟩3. QED

BY ⟨4⟩1, ⟨4⟩2

We prove that the successor history summaries, as defined in the LET of the *LL2HistorySummaryIsSuccessor* predicate, are equal to each other.

⟨3⟩13. $ll2SuccessorHistorySummary1 = ll2SuccessorHistorySummary2$

The checkpointed history summaries, as defined in the LET of the *Checkpoint* operator, are equal to each other.

⟨4⟩1. $ll2CheckpointedHistorySummary1 = ll2CheckpointedHistorySummary2$

As proven above by case analysis, the checkpointed inputs summaries, as defined in the LET of the *LL2HistorySummaryIsSuccessor* predicate, are equal.

⟨5⟩1. $ll2CheckpointedSuccessorHistorySummary1 = ll2CheckpointedSuccessorHistorySummary2$

BY ⟨3⟩12

The extension field of each successor history summary is not equal to the base hash value, because they are direct successors.

⟨5⟩2. $ll2SuccessorHistorySummary1.extension \neq BaseHashValue$

BY ⟨1⟩1, ⟨2⟩1, *SuccessorHasNonBaseExtensionLemma*DEF *ll2SuccessorHistorySummary1*

⟨5⟩3. $ll2SuccessorHistorySummary2.extension \neq BaseHashValue$

BY ⟨1⟩1, ⟨2⟩2, *SuccessorHasNonBaseExtensionLemma*DEF *ll2SuccessorHistorySummary2*

The conclusion follows from the definitions, because the Checkpoint operator is injective under the preimage constraint of an extension field that is unequal to the base hash value.

⟨5⟩4. QED

BY ⟨5⟩1, ⟨5⟩2, ⟨5⟩3

DEF *ll2CheckpointedSuccessorHistorySummary1*, *ll2CheckpointedSuccessorHistorySummary2*, *Checkpoint*, *ll2CheckpointedHistorySummary1*, *ll2CheckpointedHistorySummary2*, *checkpointedAnchor1*, *checkpointedAnchor2*

The individual fields of the successor history summaries are equal to each other, thanks to the collision resistance of the hash function.

⟨4⟩2. $\land\ ll2SuccessorHistorySummary1.anchor = ll2SuccessorHistorySummary2.anchor$

$\land\ ll2SuccessorHistorySummary1.extension = ll2SuccessorHistorySummary2.extension$

⟨5⟩1. $checkpointedAnchor1 = checkpointedAnchor2$

BY ⟨4⟩1 DEF *ll2CheckpointedHistorySummary1*, *ll2CheckpointedHistorySummary2*

⟨5⟩2. $ll2SuccessorHistorySummary1.anchor \in HashDomain$

BY ⟨3⟩5 DEF *HistorySummaryType*, *HashDomain*

⟨5⟩3. $ll2SuccessorHistorySummary1.extension \in HashDomain$

BY ⟨3⟩5 DEF *HistorySummaryType*, *HashDomain*

⟨5⟩4. *ll2SuccessorHistorySummary2.anchor* $\in$ *HashDomain*

BY ⟨3⟩9 DEF *HistorySummaryType*, *HashDomain*

⟨5⟩5. *ll2SuccessorHistorySummary2.extension* $\in$ *HashDomain*

BY ⟨3⟩9 DEF *HistorySummaryType*, *HashDomain*

⟨5⟩6. QED

Ideally, this QED step should just read:

BY ⟨5⟩1, ⟨5⟩2, ⟨5⟩3, ⟨5⟩4, ⟨5⟩5, *HashCollisionResistant* DEF *checkpointedAnchor1*, *checkpointedAnchor2*

However, the prover seems to get a little confused in this instance. We make life easier for the prover by definining some local variables and hiding their definitions before appealing to the *HashCollisionResistant* assumption.

⟨6⟩ $h1a \triangleq$ *ll2SuccessorHistorySummary1.anchor*

⟨6⟩ $h2a \triangleq$ *ll2SuccessorHistorySummary1.extension*

⟨6⟩ $h1b \triangleq$ *ll2SuccessorHistorySummary2.anchor*

⟨6⟩ $h2b \triangleq$ *ll2SuccessorHistorySummary2.extension*

⟨6⟩1. $h1a \in$ *HashDomain*

BY ⟨5⟩2

⟨6⟩2. $h2a \in$ *HashDomain*

BY ⟨5⟩3

⟨6⟩3. $h1b \in$ *HashDomain*

BY ⟨5⟩4

⟨6⟩4. $h2b \in$ *HashDomain*

BY ⟨5⟩5

⟨6⟩5. $Hash(h1a, h2a) = Hash(h1b, h2b)$

BY ⟨5⟩1 DEF *checkpointedAnchor1*, *checkpointedAnchor2*

⟨6⟩6. $h1a = h1b \land h2a = h2b$

⟨7⟩ HIDE DEF $h1a$, $h2a$, $h1b$, $h2b$

⟨7⟩1. QED

BY ⟨6⟩1, ⟨6⟩2, ⟨6⟩3, ⟨6⟩4, ⟨6⟩5, *HashCollisionResistant*

⟨6⟩7. QED

BY ⟨6⟩6

Because the fields are equal, the records are equal, but proving this requires that we prove the types of the records and invoke the *HistorySummaryRecordCompositionLemma*.

⟨4⟩3. *ll2SuccessorHistorySummary1* $\in$ *HistorySummaryType*

BY ⟨3⟩5

⟨4⟩4. *ll2SuccessorHistorySummary2* $\in$ *HistorySummaryType*

BY ⟨3⟩9

⟨4⟩5. QED

BY ⟨4⟩2, ⟨4⟩3, ⟨4⟩4, *HistorySummaryRecordCompositionLemma*DEF *HistorySummaryType*

The values that are hashed to produce the new extension value within the *Successor* operator are equal to each other, because the new extensions are equal to each other, and the hash is collision-resistant.

⟨3⟩14. $\land$ *ll2PreviousHistorySummary1.extension* = *ll2PreviousHistorySummary2.extension*

$\land$ *securedInput1* = *securedInput2*

⟨4⟩1. *newExtension1* = *newExtension2*

⟨5⟩1. *ll2NewHistorySummary1* = *ll2NewHistorySummary2*

BY ⟨3⟩13

DEF *ll2SuccessorHistorySummary1*, *ll2SuccessorHistorySummary2*, *Successor*, *ll2NewHistorySummary1*, *ll2NewHistorySummary2*, *newExtension1*, *newExtension2*, *newAnchor1*, *newAnchor2*, *securedInput1*, *securedInput2*

⟨5⟩2. QED

BY ⟨5⟩1 DEF *ll2NewHistorySummary1*, *ll2NewHistorySummary2*

⟨4⟩2. $ll2PreviousHistorySummary1.extension \in HashDomain$
BY ⟨2⟩1 DEF $HistorySummaryType$, $HashDomain$
⟨4⟩3. $securedInput1 \in HashDomain$
BY ⟨3⟩4 DEF $HashDomain$
⟨4⟩4. $ll2PreviousHistorySummary2.extension \in HashDomain$
BY ⟨2⟩2 DEF $HistorySummaryType$, $HashDomain$
⟨4⟩5. $securedInput2 \in HashDomain$
BY ⟨3⟩8 DEF $HashDomain$
⟨4⟩6. QED

Ideally, this QED step should just read:

BY ⟨4⟩1, ⟨4⟩2, ⟨4⟩3, ⟨4⟩4, ⟨4⟩5, $HashCollisionResistant$

However, the prover seems to get a little confused in this instance. We make life easier for the prover by defininining some local variables and hiding their definitions before appealing to the $HashCollisionResistant$ assumption.

⟨5⟩ $h1a \triangleq ll2PreviousHistorySummary1.extension$
⟨5⟩ $h2a \triangleq securedInput1$
⟨5⟩ $h1b \triangleq ll2PreviousHistorySummary2.extension$
⟨5⟩ $h2b \triangleq securedInput2$
⟨5⟩1. $h1a \in HashDomain$
BY ⟨4⟩2
⟨5⟩2. $h2a \in HashDomain$
BY ⟨4⟩3
⟨5⟩3. $h1b \in HashDomain$
BY ⟨4⟩4
⟨5⟩4. $h2b \in HashDomain$
BY ⟨4⟩5
⟨5⟩5. $Hash(h1a, h2a) = Hash(h1b, h2b)$
BY ⟨4⟩1 DEF $newExtension1$, $newExtension2$
⟨5⟩6. $h1a = h1b \land h2a = h2b$
⟨6⟩ HIDE DEF $h1a$, $h2a$, $h1b$, $h2b$
⟨6⟩1. QED
BY ⟨5⟩1, ⟨5⟩2, ⟨5⟩3, ⟨5⟩4, ⟨5⟩5, $HashCollisionResistant$
⟨5⟩7. QED
BY ⟨5⟩6

The previous history summaries are equal to each other, because the fields of each record are equal.

⟨3⟩15. $ll2PreviousHistorySummary1 = ll2PreviousHistorySummary2$
⟨4⟩1. $ll2PreviousHistorySummary1.anchor = ll2PreviousHistorySummary2.anchor$
⟨5⟩1. $ll2NewHistorySummary1 = ll2NewHistorySummary2$
BY ⟨3⟩13
DEF $ll2SuccessorHistorySummary1$, $ll2SuccessorHistorySummary2$, $Successor$, $ll2NewHistorySummary1$, $ll2NewHistorySummary2$, $newExtension1$, $newExtension2$, $newAnchor1$, $newAnchor2$, $securedInput1$, $securedInput2$
⟨5⟩2. QED
BY ⟨5⟩1 DEF $ll2NewHistorySummary1$, $ll2NewHistorySummary2$, $newAnchor1$, $newAnchor2$
⟨4⟩2. $ll2PreviousHistorySummary1.extension = ll2PreviousHistorySummary2.extension$
BY ⟨3⟩14

Because the fields are equal, the records are equal, but proving this requires that we prove the types of the records and invoke the $HistorySummaryRecordCompositionLemma$.

⟨4⟩3. $ll2PreviousHistorySummary1 \in HistorySummaryType$
BY ⟨2⟩1
⟨4⟩4. $ll2PreviousHistorySummary1 \in HistorySummaryType$
BY ⟨2⟩2

$\langle 4\rangle 5.$ QED
BY $\langle 4\rangle 1$, $\langle 4\rangle 2$, $\langle 4\rangle 3$, $\langle 4\rangle 4$, *HistorySummaryRecordCompositionLemma*

The input values are equal to each other, because the secured inputs are equal, and the hash function is collision-resistant.

$\langle 3\rangle 16.$ $input1 = input2$
$\langle 4\rangle 1.$ $securedInput1 = securedInput2$
BY $\langle 3\rangle 14$
$\langle 4\rangle 2.$ $hashBarrier \in HashDomain$
BY $\langle 1\rangle 1$ DEF *HashDomain*
$\langle 4\rangle 3.$ $input1 \in HashDomain$
BY $\langle 2\rangle 1$ DEF *HashDomain*
$\langle 4\rangle 4.$ $input2 \in HashDomain$
BY $\langle 2\rangle 2$ DEF *HashDomain*
$\langle 4\rangle 5.$ QED
BY $\langle 4\rangle 1$, $\langle 4\rangle 2$, $\langle 4\rangle 3$, $\langle 4\rangle 4$, *HashCollisionResistant*
DEF *securedInput1*, *securedInput2*

Each of the conjuncts is true, so the conjunction is true.

$\langle 3\rangle 17.$ QED
BY $\langle 3\rangle 15$, $\langle 3\rangle 16$

The Memoir-Basic previous history summaries are equal, because the Memoir-Opt previous history summaries are equal, and the matches are unique.

$\langle 2\rangle 4.$ $ll1PreviousHistorySummary1 = ll1PreviousHistorySummary2$
$\langle 3\rangle 1.$ $HistorySummariesMatch(ll1PreviousHistorySummary1, ll2PreviousHistorySummary1, hashBarrier)$
BY $\langle 2\rangle 1$
$\langle 3\rangle 2.$ $HistorySummariesMatch(ll1PreviousHistorySummary2, ll2PreviousHistorySummary2, hashBarrier)$
BY $\langle 2\rangle 2$
$\langle 3\rangle 3.$ $ll2PreviousHistorySummary1 = ll2PreviousHistorySummary2$
BY $\langle 2\rangle 3$
$\langle 3\rangle 4.$ QED
$\langle 4\rangle 1.$ $ll1PreviousHistorySummary1 \in HashType$
BY $\langle 2\rangle 1$
$\langle 4\rangle 2.$ $ll1PreviousHistorySummary2 \in HashType$
BY $\langle 2\rangle 2$
$\langle 4\rangle 3.$ $ll2PreviousHistorySummary1 \in HistorySummaryType$
BY $\langle 2\rangle 1$
$\langle 4\rangle 4.$ $ll2PreviousHistorySummary2 \in HistorySummaryType$
BY $\langle 2\rangle 2$
$\langle 4\rangle 5.$ $hashBarrier \in HashType$
BY $\langle 1\rangle 1$
$\langle 4\rangle 6.$ QED
BY $\langle 3\rangle 1$, $\langle 3\rangle 2$, $\langle 3\rangle 3$, $\langle 4\rangle 1$, $\langle 4\rangle 2$, $\langle 4\rangle 3$, $\langle 4\rangle 4$, $\langle 4\rangle 5$, *HistorySummariesMatchUniqueLemma*

The Memoir-Basic history summaries are equal by construction, given that the inputs to the hash functions that produce these inputs are equal.

$\langle 2\rangle 5.$ QED
$\langle 3\rangle 1.$ $ll1HistorySummary1 = Hash(ll1PreviousHistorySummary1, input1)$
BY $\langle 2\rangle 1$
$\langle 3\rangle 2.$ $ll1HistorySummary2 = Hash(ll1PreviousHistorySummary2, input2)$
BY $\langle 2\rangle 2$
$\langle 3\rangle 3.$ $input1 = input2$
BY $\langle 2\rangle 3$
$\langle 3\rangle 4.$ QED
BY $\langle 2\rangle 4$, $\langle 3\rangle 1$, $\langle 3\rangle 2$, $\langle 3\rangle 3$

The base case and the recursive case are exhaustive.

$\langle 1 \rangle 5.$ QED

BY $\langle 1 \rangle 3$, $\langle 1 \rangle 4$

4.11 Proofs of Lemmas Relating to the Memoir-Opt Implementation

MODULE *MemoirLL2ImplementationLemmas*

This module states and proves a bunch of lemmas that support the proof that the Memoir-Opt spec implements the Memoir-Basic spec.

This module includes the following theorems:

UnchangedAvailableInputsLemma
UnchangedObservedOutputsLemma
UnchangedObservedAuthenticatorsLemma
UnchangedDiskPublicStateLemma
UnchangedDiskPrivateStateEncLemma
UnchangedDiskHistorySummaryLemma
UnchangedDiskAuthenticatorLemma
UnchangedDiskLemma
UnchangedRAMPublicStateLemma
UnchangedRAMPrivateStateEncLemma
UnchangedRAMHistorySummaryLemma
UnchangedRAMAuthenticatorLemma
UnchangedRAMLemma
UnchangedNVRAMHistorySummaryLemma
UnchangedNVRAMSymmetricKeyLemma
UnchangedNVRAMLemma

EXTENDS *MemoirLL2RefinementLemmas*

The *UnchangedAvailableInputsLemma* states that when there is no change to the Memoir-Opt variable representing the available itputs, there is no change to the Memoir-Basic variable representing the available itputs.

THEOREM *UnchangedAvailableInputsLemma* $\triangleq$
$(\land$ UNCHANGED $LL2AvailableInputs$
$\land$ $LL2Refinement$
$\land$ $LL2Refinement'$
$\land$ $LL2TypeInvariant$
$\land$ $LL2TypeInvariant')$
$\Rightarrow$
UNCHANGED $LL1AvailableInputs$
$\langle 1\rangle 1.$ HAVE $\land$ UNCHANGED $LL2AvailableInputs$
$\land$ $LL2Refinement$
$\land$ $LL2Refinement'$
$\land$ $LL2TypeInvariant$
$\land$ $LL2TypeInvariant'$
$\langle 1\rangle 2.$ UNCHANGED $LL2AvailableInputs$
BY $\langle 1\rangle 1$
$\langle 1\rangle 3.$ $LL1AvailableInputs = LL2AvailableInputs$
BY $\langle 1\rangle 1$ DEF $LL2Refinement$
$\langle 1\rangle 4.$ $LL1AvailableInputs' = LL2AvailableInputs'$
BY $\langle 1\rangle 1$ DEF $LL2Refinement$
$\langle 1\rangle 5.$ QED
BY $\langle 1\rangle 2$, $\langle 1\rangle 3$, $\langle 1\rangle 4$

The *UnchangedObservedOutputsLemma* states that when there is no change to the Memoir-Opt variable representing the observed outputs, there is no change to the Memoir-Basic variable representing the observed outputs.

THEOREM $UnchangedObservedOutputsLemma \triangleq$

$$
\begin{array}{l}
\quad (\land \text{ UNCHANGED } LL2ObservedOutputs \\
\quad\ \ \land LL2Refinement \\
\quad\ \ \land LL2Refinement' \\
\quad\ \ \land LL2TypeInvariant \\
\quad\ \ \land LL2TypeInvariant') \\
\ \Rightarrow \\
\quad\ \ \text{UNCHANGED } LL1ObservedOutputs
\end{array}
$$

⟨1⟩1. HAVE $\land$ UNCHANGED $LL2ObservedOutputs$
$\land$ $LL2Refinement$
$\land$ $LL2Refinement'$
$\land$ $LL2TypeInvariant$
$\land$ $LL2TypeInvariant'$

⟨1⟩2. UNCHANGED $LL2ObservedOutputs$
BY ⟨1⟩1

⟨1⟩3. $LL1ObservedOutputs = LL2ObservedOutputs$
BY ⟨1⟩1 DEF $LL2Refinement$

⟨1⟩4. $LL1ObservedOutputs' = LL2ObservedOutputs'$
BY ⟨1⟩1 DEF $LL2Refinement$

⟨1⟩5. QED
BY ⟨1⟩2, ⟨1⟩3, ⟨1⟩4

The *UnchangedObservedAuthenticatorsLemma* states that when there is no change to the Memoir-Opt variable representing the observed authenticators, there is no change to the Memoir-Basic variable representing the observed authenticators.

THEOREM $UnchangedObservedAuthenticatorsLemma \triangleq$

$$
\begin{array}{l}
\quad (\land \text{ UNCHANGED } LL2ObservedAuthenticators \\
\quad\ \ \land \text{ UNCHANGED } LL2NVRAM.symmetricKey \\
\quad\ \ \land \text{ UNCHANGED } LL2NVRAM.hashBarrier \\
\quad\ \ \land LL2Refinement \\
\quad\ \ \land LL2Refinement' \\
\quad\ \ \land LL2TypeInvariant \\
\quad\ \ \land LL2TypeInvariant') \\
\ \Rightarrow \\
\quad\ \ \text{UNCHANGED } LL1ObservedAuthenticators
\end{array}
$$

⟨1⟩1. HAVE $\land$ UNCHANGED $LL2ObservedAuthenticators$
$\land$ UNCHANGED $LL2NVRAM.symmetricKey$
$\land$ UNCHANGED $LL2NVRAM.hashBarrier$
$\land$ $LL2Refinement$
$\land$ $LL2Refinement'$
$\land$ $LL2TypeInvariant$
$\land$ $LL2TypeInvariant'$

⟨1⟩2. UNCHANGED $LL2ObservedAuthenticators$
BY ⟨1⟩1

⟨1⟩3. UNCHANGED $LL2NVRAM.symmetricKey$
BY ⟨1⟩1

⟨1⟩4. UNCHANGED $LL2NVRAM.hashBarrier$
BY ⟨1⟩1

⟨1⟩5. $AuthenticatorSetsMatch($
$LL1ObservedAuthenticators,$

$LL2ObservedAuthenticators$,
$LL2NVRAM.symmetricKey$,
$LL2NVRAM.hashBarrier$)
BY ⟨1⟩1 DEF $LL2Refinement$
⟨1⟩6. $AuthenticatorSetsMatch($
$LL1ObservedAuthenticators'$,
$LL2ObservedAuthenticators'$,
$LL2NVRAM.symmetricKey'$,
$LL2NVRAM.hashBarrier'$)
BY ⟨1⟩1 DEF $LL2Refinement$
⟨1⟩7. QED
⟨2⟩1. $LL1ObservedAuthenticators \in$ SUBSET $MACType$
BY ⟨1⟩1 DEF $LL2Refinement$
⟨2⟩2. $LL1ObservedAuthenticators' \in$ SUBSET $MACType$
BY ⟨1⟩1 DEF $LL2Refinement$
⟨2⟩3. $LL2ObservedAuthenticators \in$ SUBSET $MACType$
BY ⟨1⟩1 DEF $LL2TypeInvariant$
⟨2⟩4. $LL2NVRAM.symmetricKey \in SymmetricKeyType$
BY ⟨1⟩1, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$
⟨2⟩5. $LL2NVRAM.hashBarrier \in HashType$
BY ⟨1⟩1, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$
⟨2⟩6. QED
BY ⟨1⟩2, ⟨1⟩3, ⟨1⟩4, ⟨1⟩5, ⟨1⟩6, ⟨2⟩1, ⟨2⟩2, ⟨2⟩3, ⟨2⟩4, ⟨2⟩5,
$AuthenticatorSetsMatchUniqueLemma$

The *UnchangedDiskPublicStateLemma* states that when there is no change to the field of the Memoir-Opt variable representing the public state in the disk, there is no change to the field of the Memoir-Basic variable representing the public state in the disk.

THEOREM $UnchangedDiskPublicStateLemma \triangleq$
($\wedge$ UNCHANGED $LL2Disk.publicState$
$\wedge$ $LL2Refinement$
$\wedge$ $LL2Refinement'$
$\wedge$ $LL2TypeInvariant$
$\wedge$ $LL2TypeInvariant'$)
$\Rightarrow$
UNCHANGED $LL1Disk.publicState$
⟨1⟩1. HAVE $\wedge$ UNCHANGED $LL2Disk.publicState$
$\wedge$ $LL2Refinement$
$\wedge$ $LL2Refinement'$
$\wedge$ $LL2TypeInvariant$
$\wedge$ $LL2TypeInvariant'$
⟨1⟩2. UNCHANGED $LL2Disk.publicState$
BY ⟨1⟩1
⟨1⟩3. $LL1Disk.publicState = LL2Disk.publicState$
BY ⟨1⟩1 DEF $LL2Refinement$
⟨1⟩4. $LL1Disk.publicState' = LL2Disk.publicState'$
BY ⟨1⟩1 DEF $LL2Refinement$
⟨1⟩5. QED
BY ⟨1⟩2, ⟨1⟩3, ⟨1⟩4

The *UnchangedDiskPrivateStateEncLemma* states that when there is no change to the field of the Memoir-Opt variable representing the encrypted private state in the disk, there is no change to the field of the Memoir-Basic variable representing the encrypted private state in the disk.

THEOREM *UnchangedDiskPrivateStateEncLemma* $\triangleq$

$(\land$ UNCHANGED $LL2Disk.privateStateEnc$
$\land$ $LL2Refinement$
$\land$ $LL2Refinement'$
$\land$ $LL2TypeInvariant$
$\land$ $LL2TypeInvariant')$
$\Rightarrow$
UNCHANGED $LL1Disk.privateStateEnc$

$\langle 1\rangle 1.$ HAVE $\land$ UNCHANGED $LL2Disk.privateStateEnc$
$\land$ $LL2Refinement$
$\land$ $LL2Refinement'$
$\land$ $LL2TypeInvariant$
$\land$ $LL2TypeInvariant'$
$\langle 1\rangle 2.$ UNCHANGED $LL2Disk.privateStateEnc$
BY $\langle 1\rangle 1$
$\langle 1\rangle 3.$ $LL1Disk.privateStateEnc = LL2Disk.privateStateEnc$
BY $\langle 1\rangle 1$ DEF $LL2Refinement$
$\langle 1\rangle 4.$ $LL1Disk.privateStateEnc' = LL2Disk.privateStateEnc'$
BY $\langle 1\rangle 1$ DEF $LL2Refinement$
$\langle 1\rangle 5.$ QED
BY $\langle 1\rangle 2$, $\langle 1\rangle 3$, $\langle 1\rangle 4$

The *UnchangedDiskHistorySummaryLemma* states that when there is no change to the field of the Memoir-Opt variable representing the history summary in the disk, there is no change to the field of the Memoir-Basic variable representing the history summary in the disk.

THEOREM *UnchangedDiskHistorySummaryLemma* $\triangleq$

$(\land$ UNCHANGED $LL2Disk.historySummary$
$\land$ UNCHANGED $LL2NVRAM.hashBarrier$
$\land$ $LL2Refinement$
$\land$ $LL2Refinement'$
$\land$ $LL2TypeInvariant$
$\land$ $LL2TypeInvariant')$
$\Rightarrow$
UNCHANGED $LL1Disk.historySummary$

$\langle 1\rangle 1.$ HAVE $\land$ UNCHANGED $LL2Disk.historySummary$
$\land$ UNCHANGED $LL2NVRAM.hashBarrier$
$\land$ $LL2Refinement$
$\land$ $LL2Refinement'$
$\land$ $LL2TypeInvariant$
$\land$ $LL2TypeInvariant'$
$\langle 1\rangle 2.$ UNCHANGED $LL2Disk.historySummary$
BY $\langle 1\rangle 1$
$\langle 1\rangle 3.$ UNCHANGED $LL2NVRAM.hashBarrier$
BY $\langle 1\rangle 1$
$\langle 1\rangle 4.$ $HistorySummariesMatch($
$LL1Disk.historySummary,$

$LL2Disk.historySummary$,
$LL2NVRAM.hashBarrier$)
BY ⟨1⟩1 DEF *LL2Refinement*
⟨1⟩5. *HistorySummariesMatch*(
$LL1Disk.historySummary'$,
$LL2Disk.historySummary'$,
$LL2NVRAM.hashBarrier'$)
BY ⟨1⟩1 DEF *LL2Refinement*
⟨1⟩6. QED
⟨2⟩1. $LL1Disk.historySummary \in HashType$
BY ⟨1⟩1 DEF *LL2Refinement*, *LL1UntrustedStorageType*
⟨2⟩2. $LL1Disk.historySummary' \in HashType$
BY ⟨1⟩1 DEF *LL2Refinement*, *LL1UntrustedStorageType*
⟨2⟩3. $LL2Disk.historySummary \in HistorySummaryType$
BY ⟨1⟩1, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*
⟨2⟩4. $LL2NVRAM.hashBarrier \in HashType$
BY ⟨1⟩1, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*
⟨2⟩5. QED
BY ⟨1⟩2, ⟨1⟩3, ⟨1⟩4, ⟨1⟩5, ⟨2⟩1, ⟨2⟩2, ⟨2⟩3, ⟨2⟩4,
HistorySummariesMatchUniqueLemma

The *UnchangedDiskAuthenticatorLemma* states that when there is no change to the field of the Memoir-Opt variable representing the authenticator in the disk, there is no change to the field of the Memoir-Basic variable representing the authenticator in the disk.

THEOREM *UnchangedDiskAuthenticatorLemma* $\triangleq$
($\land$ UNCHANGED $LL2Disk.authenticator$
$\land$ UNCHANGED $LL2NVRAM.hashBarrier$
$\land$ *LL2Refinement*
$\land$ *LL2Refinement*′
$\land$ *LL2TypeInvariant*
$\land$ *LL2TypeInvariant*′)
$\Rightarrow$
UNCHANGED $LL1Disk.authenticator$
⟨1⟩1. HAVE $\land$ UNCHANGED $LL2Disk.authenticator$
$\land$ UNCHANGED $LL2NVRAM.hashBarrier$
$\land$ *LL2Refinement*
$\land$ *LL2Refinement*′
$\land$ *LL2TypeInvariant*
$\land$ *LL2TypeInvariant*′
⟨1⟩2. UNCHANGED $LL2Disk.authenticator$
BY ⟨1⟩1
⟨1⟩3. UNCHANGED $LL2NVRAM.hashBarrier$
BY ⟨1⟩1
⟨1⟩4. $\exists\, symmetricKey \in SymmetricKeyType$:
AuthenticatorsMatch(
$LL1Disk.authenticator$,
$LL2Disk.authenticator$,
$symmetricKey$,
$LL2NVRAM.hashBarrier$)
BY ⟨1⟩1 DEF *LL2Refinement*

$\langle 1 \rangle 5.\ \exists\, symmetricKey \in SymmetricKeyType :$
$\quad AuthenticatorsMatch($
$\quad\quad LL1Disk.authenticator',$
$\quad\quad LL2Disk.authenticator',$
$\quad\quad symmetricKey,$
$\quad\quad LL2NVRAM.hashBarrier')$
BY $\langle 1 \rangle 1$ DEF $LL2Refinement$
$\langle 1 \rangle 6.$ QED
$\langle 2 \rangle 1.\ LL1Disk.authenticator \in MACType$
BY $\langle 1 \rangle 1$ DEF $LL2Refinement,\ LL1UntrustedStorageType$
$\langle 2 \rangle 2.\ LL1Disk.authenticator' \in MACType$
BY $\langle 1 \rangle 1$ DEF $LL2Refinement,\ LL1UntrustedStorageType$
$\langle 2 \rangle 3.\ LL2Disk.authenticator \in MACType$
BY $\langle 1 \rangle 1,\ LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$
$\langle 2 \rangle 4.\ LL2NVRAM.hashBarrier \in HashType$
BY $\langle 1 \rangle 1,\ LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$
$\langle 2 \rangle 5.$ QED
BY $\langle 1 \rangle 2,\ \langle 1 \rangle 3,\ \langle 1 \rangle 4,\ \langle 1 \rangle 5,\ \langle 2 \rangle 1,\ \langle 2 \rangle 2,\ \langle 2 \rangle 3,\ \langle 2 \rangle 4,$
$AuthenticatorsMatchUniqueLemma$

The *UnchangedDiskLemma* states that when there is no change to the Memoir-Opt variable representing the disk, there is no change to the Memoir-Basic variable representing the disk.

THEOREM $UnchangedDiskLemma \triangleq$
$(\land$ UNCHANGED $LL2Disk$
$\land$ UNCHANGED $LL2NVRAM.symmetricKey$
$\land$ UNCHANGED $LL2NVRAM.hashBarrier$
$\land\ LL2Refinement$
$\land\ LL2Refinement'$
$\land\ LL2TypeInvariant$
$\land\ LL2TypeInvariant')$
$\Rightarrow$
UNCHANGED $LL1Disk$
$\langle 1 \rangle 1.$ HAVE $\land$ UNCHANGED $LL2Disk$
$\land$ UNCHANGED $LL2NVRAM.symmetricKey$
$\land$ UNCHANGED $LL2NVRAM.hashBarrier$
$\land\ LL2Refinement$
$\land\ LL2Refinement'$
$\land\ LL2TypeInvariant$
$\land\ LL2TypeInvariant'$
$\langle 1 \rangle 2.$ UNCHANGED $LL1Disk.publicState$
$\langle 2 \rangle 1.$ UNCHANGED $LL2Disk.publicState$
BY $\langle 1 \rangle 1$
$\langle 2 \rangle 2.$ QED
BY $\langle 1 \rangle 1,\ \langle 2 \rangle 1,\ UnchangedDiskPublicStateLemma$
$\langle 1 \rangle 3.$ UNCHANGED $LL1Disk.privateStateEnc$
$\langle 2 \rangle 1.$ UNCHANGED $LL2Disk.privateStateEnc$
BY $\langle 1 \rangle 1$
$\langle 2 \rangle 2.$ QED
BY $\langle 1 \rangle 1,\ \langle 2 \rangle 1,\ UnchangedDiskPrivateStateEncLemma$
$\langle 1 \rangle 4.$ UNCHANGED $LL1Disk.historySummary$

$\langle 2 \rangle 1.$ UNCHANGED $LL2Disk.historySummary$
BY $\langle 1 \rangle 1$
$\langle 2 \rangle 2.$ QED
BY $\langle 1 \rangle 1$, $\langle 2 \rangle 1$, $UnchangedDiskHistorySummaryLemma$
$\langle 1 \rangle 5.$ UNCHANGED $LL1Disk.authenticator$
$\langle 2 \rangle 1.$ UNCHANGED $LL2Disk.authenticator$
BY $\langle 1 \rangle 1$
$\langle 2 \rangle 2.$ QED
BY $\langle 1 \rangle 1$, $\langle 2 \rangle 1$, $UnchangedDiskAuthenticatorLemma$
$\langle 1 \rangle 6.$ $LL1Disk \in LL1UntrustedStorageType$
BY $\langle 1 \rangle 1$ DEF $LL2Refinement$
$\langle 1 \rangle 7.$ $LL1Disk' \in LL1UntrustedStorageType$
BY $\langle 1 \rangle 1$ DEF $LL2Refinement$
$\langle 1 \rangle 8.$ QED
BY $\langle 1 \rangle 2$, $\langle 1 \rangle 3$, $\langle 1 \rangle 4$, $\langle 1 \rangle 5$, $\langle 1 \rangle 6$, $\langle 1 \rangle 7$, $LL1DiskRecordCompositionLemma$

The *UnchangedRAMPublicStateLemma* states that when there is no change to the field of the Memoir-Opt variable representing the public state in the RAM, there is no change to the field of the Memoir-Basic variable representing the public state in the RAM.

THEOREM $UnchangedRAMPublicStateLemma \triangleq$
$(\land$ UNCHANGED $LL2RAM.publicState$
$\land\ LL2Refinement$
$\land\ LL2Refinement'$
$\land\ LL2TypeInvariant$
$\land\ LL2TypeInvariant')$
$\Rightarrow$
UNCHANGED $LL1RAM.publicState$
$\langle 1 \rangle 1.$ HAVE $\land$ UNCHANGED $LL2RAM.publicState$
$\land\ LL2Refinement$
$\land\ LL2Refinement'$
$\land\ LL2TypeInvariant$
$\land\ LL2TypeInvariant'$
$\langle 1 \rangle 2.$ UNCHANGED $LL2RAM.publicState$
BY $\langle 1 \rangle 1$
$\langle 1 \rangle 3.$ $LL1RAM.publicState = LL2RAM.publicState$
BY $\langle 1 \rangle 1$ DEF $LL2Refinement$
$\langle 1 \rangle 4.$ $LL1RAM.publicState' = LL2RAM.publicState'$
BY $\langle 1 \rangle 1$ DEF $LL2Refinement$
$\langle 1 \rangle 5.$ QED
BY $\langle 1 \rangle 2$, $\langle 1 \rangle 3$, $\langle 1 \rangle 4$

The *UnchangedRAMPrivateStateEncLemma* states that when there is no change to the field of the Memoir-Opt variable representing the encrypted private state in the RAM, there is no change to the field of the Memoir-Basic variable representing the encrypted private state in the RAM.

THEOREM $UnchangedRAMPrivateStateEncLemma \triangleq$
$(\land$ UNCHANGED $LL2RAM.privateStateEnc$
$\land\ LL2Refinement$
$\land\ LL2Refinement'$

$\land$ *LL2TypeInvariant*
$\land$ *LL2TypeInvariant'*)
$\Rightarrow$
UNCHANGED *LL1RAM.privateStateEnc*
⟨1⟩1. HAVE $\land$ UNCHANGED *LL2RAM.privateStateEnc*
$\land$ *LL2Refinement*
$\land$ *LL2Refinement'*
$\land$ *LL2TypeInvariant*
$\land$ *LL2TypeInvariant'*
⟨1⟩2. UNCHANGED *LL2RAM.privateStateEnc*
BY ⟨1⟩1
⟨1⟩3. *LL1RAM.privateStateEnc* = *LL2RAM.privateStateEnc*
BY ⟨1⟩1 DEF *LL2Refinement*
⟨1⟩4. *LL1RAM.privateStateEnc'* = *LL2RAM.privateStateEnc'*
BY ⟨1⟩1 DEF *LL2Refinement*
⟨1⟩5. QED
BY ⟨1⟩2, ⟨1⟩3, ⟨1⟩4

The *UnchangedRAMHistorySummaryLemma* states that when there is no change to the field of the Memoir-Opt variable representing the history summary in the RAM, there is no change to the field of the Memoir-Basic variable representing the history summary in the RAM.

THEOREM *UnchangedRAMHistorySummaryLemma* $\triangleq$
($\land$ UNCHANGED *LL2RAM.historySummary*
$\land$ UNCHANGED *LL2NVRAM.hashBarrier*
$\land$ *LL2Refinement*
$\land$ *LL2Refinement'*
$\land$ *LL2TypeInvariant*
$\land$ *LL2TypeInvariant'*)
$\Rightarrow$
UNCHANGED *LL1RAM.historySummary*
⟨1⟩1. HAVE $\land$ UNCHANGED *LL2RAM.historySummary*
$\land$ UNCHANGED *LL2NVRAM.hashBarrier*
$\land$ *LL2Refinement*
$\land$ *LL2Refinement'*
$\land$ *LL2TypeInvariant*
$\land$ *LL2TypeInvariant'*
⟨1⟩2. UNCHANGED *LL2RAM.historySummary*
BY ⟨1⟩1
⟨1⟩3. UNCHANGED *LL2NVRAM.hashBarrier*
BY ⟨1⟩1
⟨1⟩4. *HistorySummariesMatch*(
LL1RAM.historySummary,
LL2RAM.historySummary,
LL2NVRAM.hashBarrier)
BY ⟨1⟩1 DEF *LL2Refinement*
⟨1⟩5. *HistorySummariesMatch*(
LL1RAM.historySummary',
LL2RAM.historySummary',
LL2NVRAM.hashBarrier')
BY ⟨1⟩1 DEF *LL2Refinement*

⟨1⟩6. QED
  ⟨2⟩1. $LL1RAM.historySummary \in HashType$
    BY ⟨1⟩1 DEF $LL2Refinement$, $LL1UntrustedStorageType$
  ⟨2⟩2. $LL1RAM.historySummary' \in HashType$
    BY ⟨1⟩1 DEF $LL2Refinement$, $LL1UntrustedStorageType$
  ⟨2⟩3. $LL2RAM.historySummary \in HistorySummaryType$
    BY ⟨1⟩1, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$
  ⟨2⟩4. $LL2NVRAM.hashBarrier \in HashType$
    BY ⟨1⟩1, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$
  ⟨2⟩5. QED
    BY ⟨1⟩2, ⟨1⟩3, ⟨1⟩4, ⟨1⟩5, ⟨2⟩1, ⟨2⟩2, ⟨2⟩3, ⟨2⟩4,
    $HistorySummariesMatchUniqueLemma$

The *UnchangedRAMAuthenticatorLemma* states that when there is no change to the field of the Memoir-Opt variable representing the authenticator in the RAM, there is no change to the field of the Memoir-Basic variable representing the authenticator in the RAM.

THEOREM $UnchangedRAMAuthenticatorLemma \triangleq$
  ($\land$ UNCHANGED $LL2RAM.authenticator$
    $\land$ UNCHANGED $LL2NVRAM.hashBarrier$
    $\land$ $LL2Refinement$
    $\land$ $LL2Refinement'$
    $\land$ $LL2TypeInvariant$
    $\land$ $LL2TypeInvariant'$)
  $\Rightarrow$
    UNCHANGED $LL1RAM.authenticator$
⟨1⟩1. HAVE $\land$ UNCHANGED $LL2RAM.authenticator$
  $\land$ UNCHANGED $LL2NVRAM.hashBarrier$
  $\land$ $LL2Refinement$
  $\land$ $LL2Refinement'$
  $\land$ $LL2TypeInvariant$
  $\land$ $LL2TypeInvariant'$
⟨1⟩2. UNCHANGED $LL2RAM.authenticator$
  BY ⟨1⟩1
⟨1⟩3. UNCHANGED $LL2NVRAM.hashBarrier$
  BY ⟨1⟩1
⟨1⟩4. $\exists\, symmetricKey \in SymmetricKeyType$:
    $AuthenticatorsMatch($
      $LL1RAM.authenticator,$
      $LL2RAM.authenticator,$
      $symmetricKey,$
      $LL2NVRAM.hashBarrier)$
  BY ⟨1⟩1 DEF $LL2Refinement$
⟨1⟩5. $\exists\, symmetricKey \in SymmetricKeyType$:
    $AuthenticatorsMatch($
      $LL1RAM.authenticator',$
      $LL2RAM.authenticator',$
      $symmetricKey,$
      $LL2NVRAM.hashBarrier')$
  BY ⟨1⟩1 DEF $LL2Refinement$
⟨1⟩6. QED

$\langle 2\rangle 1.$ $LL1RAM.authenticator \in MACType$
BY $\langle 1\rangle 1$ DEF $LL2Refinement$, $LL1UntrustedStorageType$
$\langle 2\rangle 2.$ $LL1RAM.authenticator' \in MACType$
BY $\langle 1\rangle 1$ DEF $LL2Refinement$, $LL1UntrustedStorageType$
$\langle 2\rangle 3.$ $LL2RAM.authenticator \in MACType$
BY $\langle 1\rangle 1$, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$
$\langle 2\rangle 4.$ $LL2NVRAM.hashBarrier \in HashType$
BY $\langle 1\rangle 1$, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$
$\langle 2\rangle 5.$ QED
BY $\langle 1\rangle 2$, $\langle 1\rangle 3$, $\langle 1\rangle 4$, $\langle 1\rangle 5$, $\langle 2\rangle 1$, $\langle 2\rangle 2$, $\langle 2\rangle 3$, $\langle 2\rangle 4$, $AuthenticatorsMatchUniqueLemma$

The *UnchangedRAMLemma* states that when there is no change to the Memoir-Opt variable representing the RAM, there is no change to the Memoir-Basic variable representing the RAM.

THEOREM $UnchangedRAMLemma \triangleq$
$(\land$ UNCHANGED $LL2RAM$
$\land$ UNCHANGED $LL2NVRAM.symmetricKey$
$\land$ UNCHANGED $LL2NVRAM.hashBarrier$
$\land$ $LL2Refinement$
$\land$ $LL2Refinement'$
$\land$ $LL2TypeInvariant$
$\land$ $LL2TypeInvariant')$
$\Rightarrow$
UNCHANGED $LL1RAM$
$\langle 1\rangle 1.$ HAVE $\land$ UNCHANGED $LL2RAM$
$\land$ UNCHANGED $LL2NVRAM.symmetricKey$
$\land$ UNCHANGED $LL2NVRAM.hashBarrier$
$\land$ $LL2Refinement$
$\land$ $LL2Refinement'$
$\land$ $LL2TypeInvariant$
$\land$ $LL2TypeInvariant'$
$\langle 1\rangle 2.$ UNCHANGED $LL1RAM.publicState$
$\langle 2\rangle 1.$ UNCHANGED $LL2RAM.publicState$
BY $\langle 1\rangle 1$
$\langle 2\rangle 2.$ QED
BY $\langle 1\rangle 1$, $\langle 2\rangle 1$, $UnchangedRAMPublicStateLemma$
$\langle 1\rangle 3.$ UNCHANGED $LL1RAM.privateStateEnc$
$\langle 2\rangle 1.$ UNCHANGED $LL2RAM.privateStateEnc$
BY $\langle 1\rangle 1$
$\langle 2\rangle 2.$ QED
BY $\langle 1\rangle 1$, $\langle 2\rangle 1$, $UnchangedRAMPrivateStateEncLemma$
$\langle 1\rangle 4.$ UNCHANGED $LL1RAM.historySummary$
$\langle 2\rangle 1.$ UNCHANGED $LL2RAM.historySummary$
BY $\langle 1\rangle 1$
$\langle 2\rangle 2.$ QED
BY $\langle 1\rangle 1$, $\langle 2\rangle 1$, $UnchangedRAMHistorySummaryLemma$
$\langle 1\rangle 5.$ UNCHANGED $LL1RAM.authenticator$
$\langle 2\rangle 1.$ UNCHANGED $LL2RAM.authenticator$
BY $\langle 1\rangle 1$
$\langle 2\rangle 2.$ QED

BY $\langle 1\rangle 1$, $\langle 2\rangle 1$, $\mathit{UnchangedRAMAuthenticatorLemma}$
$\langle 1\rangle 6$. $\mathit{LL1RAM} \in \mathit{LL1UntrustedStorageType}$
BY $\langle 1\rangle 1$ DEF $\mathit{LL2Refinement}$
$\langle 1\rangle 7$. $\mathit{LL1RAM}' \in \mathit{LL1UntrustedStorageType}$
BY $\langle 1\rangle 1$ DEF $\mathit{LL2Refinement}$
$\langle 1\rangle 8$. QED
BY $\langle 1\rangle 2$, $\langle 1\rangle 3$, $\langle 1\rangle 4$, $\langle 1\rangle 5$, $\langle 1\rangle 6$, $\langle 1\rangle 7$, $\mathit{LL1RAMRecordCompositionLemma}$

The *UnchangedNVRAMHistorySummaryLemma* states that when there is no change to the *LL2NVRAMLogicalHistorySummary* value, which represents the logical value of the history summary in the *NVRAM* and *SPCR* of the Memoir-Opt spec, there is no change to the field of the Memoir-Basic variable representing the history summary in the *NVRAM*.

THEOREM $\mathit{UnchangedNVRAMHistorySummaryLemma} \triangleq$
($\land$ UNCHANGED $\mathit{LL2NVRAMLogicalHistorySummary}$
$\land$ UNCHANGED $\mathit{LL2NVRAM.symmetricKey}$
$\land$ UNCHANGED $\mathit{LL2NVRAM.hashBarrier}$
$\land$ $\mathit{LL2Refinement}$
$\land$ $\mathit{LL2Refinement}'$
$\land$ $\mathit{LL2TypeInvariant}$
$\land$ $\mathit{LL2TypeInvariant}'$)
$\Rightarrow$
UNCHANGED $\mathit{LL1NVRAM.historySummary}$
$\langle 1\rangle 1$. HAVE $\land$ UNCHANGED $\mathit{LL2NVRAMLogicalHistorySummary}$
$\land$ UNCHANGED $\mathit{LL2NVRAM.symmetricKey}$
$\land$ UNCHANGED $\mathit{LL2NVRAM.hashBarrier}$
$\land$ $\mathit{LL2Refinement}$
$\land$ $\mathit{LL2Refinement}'$
$\land$ $\mathit{LL2TypeInvariant}$
$\land$ $\mathit{LL2TypeInvariant}'$
$\langle 1\rangle 2$. UNCHANGED $\mathit{LL2NVRAMLogicalHistorySummary}$
BY $\langle 1\rangle 1$
$\langle 1\rangle 3$. UNCHANGED $\mathit{LL2NVRAM.symmetricKey}$
BY $\langle 1\rangle 1$
$\langle 1\rangle 4$. UNCHANGED $\mathit{LL2NVRAM.hashBarrier}$
BY $\langle 1\rangle 1$
$\langle 1\rangle 5$. $\mathit{HistorySummariesMatch}($
$\mathit{LL1NVRAM.historySummary}$,
$\mathit{LL2NVRAMLogicalHistorySummary}$,
$\mathit{LL2NVRAM.hashBarrier})$
BY $\langle 1\rangle 1$ DEF $\mathit{LL2Refinement}$
$\langle 1\rangle 6$. $\mathit{HistorySummariesMatch}($
$\mathit{LL1NVRAM.historySummary}'$,
$\mathit{LL2NVRAMLogicalHistorySummary}'$,
$\mathit{LL2NVRAM.hashBarrier}')$
BY $\langle 1\rangle 1$ DEF $\mathit{LL2Refinement}$
$\langle 1\rangle 7$. QED
$\langle 2\rangle 1$. $\mathit{LL1NVRAM.historySummary} \in \mathit{HashType}$
BY $\langle 1\rangle 1$ DEF $\mathit{LL2Refinement}$, $\mathit{LL1TrustedStorageType}$
$\langle 2\rangle 2$. $\mathit{LL1NVRAM.historySummary}' \in \mathit{HashType}$
BY $\langle 1\rangle 1$ DEF $\mathit{LL2Refinement}$, $\mathit{LL1TrustedStorageType}$

$\langle 2 \rangle 3.$ $LL2NVRAMLogicalHistorySummary \in HistorySummaryType$
BY $\langle 1 \rangle 1$, $LL2NVRAMLogicalHistorySummaryTypeSafe$
$\langle 2 \rangle 4.$ $LL2NVRAM.hashBarrier \in HashType$
BY $\langle 1 \rangle 1$, $LL2SubtypeImplicationLemma$ DEF $LL2SubtypeImplication$
$\langle 2 \rangle 5.$ QED
BY $\langle 1 \rangle 2$, $\langle 1 \rangle 3$, $\langle 1 \rangle 4$, $\langle 1 \rangle 5$, $\langle 1 \rangle 6$, $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, $\langle 2 \rangle 3$, $\langle 2 \rangle 4$,
$HistorySummariesMatchUniqueLemma$

The *UnchangedNVRAMSymmetricKeyLemma* states that when there is no change to the field of the Memoir-Opt variable representing the symmetric key in the RAM, there is no change to the field of the Memoir-Basic variable representing the symmetric key in the RAM.

THEOREM $UnchangedNVRAMSymmetricKeyLemma \triangleq$
$(\land$ UNCHANGED $LL2NVRAM.symmetricKey$
$\land\ LL2Refinement$
$\land\ LL2Refinement'$
$\land\ LL2TypeInvariant$
$\land\ LL2TypeInvariant')$
$\Rightarrow$
UNCHANGED $LL1NVRAM.symmetricKey$
$\langle 1 \rangle 1.$ HAVE $\land$ UNCHANGED $LL2NVRAM.symmetricKey$
$\land\ LL2Refinement$
$\land\ LL2Refinement'$
$\land\ LL2TypeInvariant$
$\land\ LL2TypeInvariant'$
$\langle 1 \rangle 2.$ UNCHANGED $LL2NVRAM.symmetricKey$
BY $\langle 1 \rangle 1$
$\langle 1 \rangle 3.$ $LL1NVRAM.symmetricKey = LL2NVRAM.symmetricKey$
BY $\langle 1 \rangle 1$ DEF $LL2Refinement$
$\langle 1 \rangle 4.$ $LL1NVRAM.symmetricKey' = LL2NVRAM.symmetricKey'$
BY $\langle 1 \rangle 1$ DEF $LL2Refinement$
$\langle 1 \rangle 5.$ QED
BY $\langle 1 \rangle 2$, $\langle 1 \rangle 3$, $\langle 1 \rangle 4$

The *UnchangedNVRAMLemma* states that when there is no change to the Memoir-Opt variables representing the *NVRAM* and *SPCR*, there is no change to the Memoir-Basic variable representing the *NVRAM*.

THEOREM $UnchangedNVRAMLemma \triangleq$
$(\land$ UNCHANGED $LL2NVRAM$
$\land$ UNCHANGED $LL2SPCR$
$\land\ LL2Refinement$
$\land\ LL2Refinement'$
$\land\ LL2TypeInvariant$
$\land\ LL2TypeInvariant')$
$\Rightarrow$
UNCHANGED $LL1NVRAM$
$\langle 1 \rangle 1.$ HAVE $\land$ UNCHANGED $LL2NVRAM$
$\land$ UNCHANGED $LL2SPCR$
$\land\ LL2Refinement$
$\land\ LL2Refinement'$

$\qquad\qquad\wedge\ \ LL2TypeInvariant$

$\qquad\qquad\wedge\ \ LL2TypeInvariant'$

$\langle 1\rangle 2.$ UNCHANGED $LL1NVRAM.historySummary$

$\quad\langle 2\rangle 1.$ UNCHANGED $LL2NVRAMLogicalHistorySummary$

$\qquad$ BY $\langle 1\rangle 1$ DEF $LL2NVRAMLogicalHistorySummary$

$\quad\langle 2\rangle 2.$ QED

$\qquad$ BY $\langle 1\rangle 1$, $\langle 2\rangle 1$, $UnchangedNVRAMHistorySummaryLemma$

$\langle 1\rangle 3.$ UNCHANGED $LL1NVRAM.symmetricKey$

$\quad\langle 2\rangle 1.$ UNCHANGED $LL2NVRAM.symmetricKey$

$\qquad$ BY $\langle 1\rangle 1$

$\quad\langle 2\rangle 2.$ QED

$\qquad$ BY $\langle 1\rangle 1$, $\langle 2\rangle 1$, $UnchangedNVRAMSymmetricKeyLemma$

$\langle 1\rangle 4.$ $LL1NVRAM \in LL1TrustedStorageType$

$\quad$ BY $\langle 1\rangle 1$ DEF $LL2Refinement$

$\langle 1\rangle 5.$ $LL1NVRAM' \in LL1TrustedStorageType$

$\quad$ BY $\langle 1\rangle 1$ DEF $LL2Refinement$

$\langle 1\rangle 6.$ QED

$\quad$ BY $\langle 1\rangle 2$, $\langle 1\rangle 3$, $\langle 1\rangle 4$, $\langle 1\rangle 5$, $LL1NVRAMRecordCompositionLemma$

4.12 Proof that Memoir-Opt Spec Implements Memoir-Basic Spec

MODULE *MemoirLL2Implementation*

This module proves that the Memoir-Opt spec implements the Memoir-Basic spec, under the defined refinement.

EXTENDS *MemoirLL2ImplementationLemmas*

The *LL2Implementation* theorem is where the rubber meets the road. This is the ultimate proof that the Memoir-Opt spec implements the Memoir-Basic spec, under the defined refinement.

THEOREM $LL2Implementation \triangleq LL2Spec \land \Box LL2Refinement \Rightarrow LL1Spec$

This proof will require the *LL2TypeInvariant*. Fortunately, the *LL2TypeSafe* theorem has already proven that the Memoir-Opt spec satisfies its type invariant.

$\langle 1 \rangle 1.$ $LL2Spec \Rightarrow \Box LL2TypeInvariant$
BY $LL2TypeSafe$

The top level of the proof is boilerplate TLA+ for a *StepSimulation* proof. First, we prove that the initial predicate of the Memoir-Opt spec, conjoined with the *LL2Refinement* and type invariant, implies the initial predicate of the Memoir-Basic spec. Second, we prove that the *LL2Next* predicate, conjoined with the *LL2Refinement* and type invariant in both primed and unprimed states, implies the *LL1Next* predicate. Third, we use temporal induction to prove that these two conditions imply that, if the *LL2Refinement* and the type invariant always hold, the *LL2Spec* implies the *LL1Spec*.

$\langle 1 \rangle 2.$ $LL2Init \land LL2Refinement \land LL2TypeInvariant \Rightarrow LL1Init$

We begin the base case by assuming the antecedent.

$\langle 2 \rangle 1.$ HAVE $LL2Init \land LL2Refinement \land LL2TypeInvariant$

We pick a symmetric key and a hash barrier that satisfy the *LL2Init* predicate.

$\langle 2 \rangle 2.$ PICK $symmetricKey \in SymmetricKeyType,\ hashBarrier \in HashType :$
$LL2Init!(symmetricKey,\ hashBarrier)!1$
BY $\langle 2 \rangle 1$ DEF $LL2Init$

We re-state the definitions from *LL2Init*.

$\langle 2 \rangle$ $initialPrivateStateEnc \triangleq SymmetricEncrypt(symmetricKey,\ InitialPrivateState)$
$\langle 2 \rangle$ $initialStateHash \triangleq Hash(InitialPublicState,\ initialPrivateStateEnc)$
$\langle 2 \rangle$ $ll2InitialHistorySummary \triangleq [$
$anchor \mapsto BaseHashValue,$
$extension \mapsto BaseHashValue]$
$\langle 2 \rangle$ $ll2InitialHistorySummaryHash \triangleq Hash(BaseHashValue,\ BaseHashValue)$
$\langle 2 \rangle$ $ll2InitialHistoryStateBinding \triangleq Hash(ll2InitialHistorySummaryHash,\ initialStateHash)$
$\langle 2 \rangle$ $ll2InitialAuthenticator \triangleq GenerateMAC(symmetricKey,\ ll2InitialHistoryStateBinding)$
$\langle 2 \rangle$ $ll2InitialUntrustedStorage \triangleq [$
$publicState \mapsto InitialPublicState,$
$privateStateEnc \mapsto initialPrivateStateEnc,$
$historySummary \mapsto ll2InitialHistorySummary,$
$authenticator \mapsto ll2InitialAuthenticator]$
$\langle 2 \rangle$ $ll2InitialTrustedStorage \triangleq [$
$historySummaryAnchor \mapsto BaseHashValue,$
$symmetricKey \mapsto symmetricKey,$
$hashBarrier \mapsto hashBarrier,$
$extensionInProgress \mapsto \text{FALSE}]$

We prove that the definitions from *LL2Init* satisfy their types, using the *LL2InitDefsTypeSafeLemma*.

$\langle 2 \rangle 3.$ $\land\ initialPrivateStateEnc \in PrivateStateEncType$
$\land\ initialStateHash \in HashType$
$\land\ ll2InitialHistorySummary \in HistorySummaryType$
$\land\ ll2InitialHistorySummaryHash \in HashType$
$\land\ ll2InitialHistoryStateBinding \in HashType$

$\wedge$ *ll2InitialAuthenticator* $\in$ *MACType*
$\wedge$ *ll2InitialUntrustedStorage* $\in$ *LL2UntrustedStorageType*
$\wedge$ *ll2InitialTrustedStorage* $\in$ *LL2TrustedStorageType*
$\langle 3\rangle$1. *symmetricKey* $\in$ *SymmetricKeyType*
BY $\langle 2\rangle$2
$\langle 3\rangle$2. QED
BY $\langle 3\rangle$1, *LL2InitDefsTypeSafeLemma*

We hide the definitions from *LL2Init*.

$\langle 2\rangle$ HIDE DEF *initialPrivateStateEnc*, *initialStateHash*, *ll2InitialHistorySummary*, *ll2InitialHistorySummaryHash*, *ll2InitialHistoryStateBinding*, *ll2InitialAuthenticator*, *ll2InitialUntrustedStorage*, *ll2InitialTrustedStorage*

We re-state the definitions from *LL1Init*, except for *initialPrivateStateEnc* and *initialStateHash*, which are the same as the definitions from *LL2Init*.

$\langle 2\rangle$ *ll1InitialHistoryStateBinding* $\triangleq$ *Hash*(*BaseHashValue*, *initialStateHash*)
$\langle 2\rangle$ *ll1InitialAuthenticator* $\triangleq$ *GenerateMAC*(*symmetricKey*, *ll1InitialHistoryStateBinding*)
$\langle 2\rangle$ *ll1InitialUntrustedStorage* $\triangleq$ [
publicState $\mapsto$ *InitialPublicState*,
privateStateEnc $\mapsto$ *initialPrivateStateEnc*,
historySummary $\mapsto$ *BaseHashValue*,
authenticator $\mapsto$ *ll1InitialAuthenticator*]
$\langle 2\rangle$ *ll1InitialTrustedStorage* $\triangleq$ [
historySummary $\mapsto$ *BaseHashValue*,
symmetricKey $\mapsto$ *symmetricKey*]

We prove that the definitions from *LL1Init* satisfy their types, using the *LL1InitDefsTypeSafeLemma*.

$\langle 2\rangle$4. $\wedge$ *ll1InitialHistoryStateBinding* $\in$ *HashType*
$\wedge$ *ll1InitialAuthenticator* $\in$ *MACType*
$\wedge$ *ll1InitialUntrustedStorage* $\in$ *LL1UntrustedStorageType*
$\wedge$ *ll1InitialTrustedStorage* $\in$ *LL1TrustedStorageType*
$\langle 3\rangle$1. *symmetricKey* $\in$ *SymmetricKeyType*
BY $\langle 2\rangle$2
$\langle 3\rangle$2. QED
BY $\langle 3\rangle$1, *LL1InitDefsTypeSafeLemma*DEF *initialPrivateStateEnc*, *initialStateHash*

We hide the definitions from *LL1Init*.

$\langle 2\rangle$ HIDE DEF *ll1InitialHistoryStateBinding*, *ll1InitialAuthenticator*, *ll1InitialUntrustedStorage*, *ll1InitialTrustedStorage*

One fact that will be useful in several places is that the initial history summaries match across the two specs. This follows fairly simply from the definition of *HistorySummariesMatch*.

$\langle 2\rangle$5. *HistorySummariesMatch*(*BaseHashValue*, *ll2InitialHistorySummary*, *LL2NVRAM.hashBarrier*)
$\langle 3\rangle$1. *BaseHashValue* $\in$ *HashType*
BY *ConstantsTypeSafe*
$\langle 3\rangle$2. *ll2InitialHistorySummary* $\in$ *HistorySummaryType*
BY *ConstantsTypeSafe*, *BaseHashValueTypeSafe*DEF *ll2InitialHistorySummary*, *HistorySummaryType*
$\langle 3\rangle$3. *LL2NVRAM.hashBarrier* $\in$ *HashType*
BY $\langle 2\rangle$1, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*
$\langle 3\rangle$4. QED
BY $\langle 3\rangle$1, $\langle 3\rangle$2, $\langle 3\rangle$3, *HistorySummariesMatchDefinition*DEF *ll2InitialHistorySummary*

The most interesting part of the proof for the initial state is proving that the corresponding initial authenticator values match across the two specs. We will use this in three places below: for *LL1Disk.authenticator*, *LL1RAM.authenticator*, and *LL1ObservedAuthenticators*. We prove the match using the definition of the *AuthenticatorsMatch* predicate.

$\langle 2\rangle$6. *AuthenticatorsMatch*(
ll1InitialAuthenticator,

$ll2InitialAuthenticator$,
$LL2NVRAM.symmetricKey$,
$LL2NVRAM.hashBarrier$)

First, we prove some types needed by the definition of the *AuthenticatorsMatch* predicate.

⟨3⟩1. $initialStateHash \in HashType$
BY ⟨2⟩3
⟨3⟩2. $BaseHashValue \in HashType$
BY $ConstantsTypeSafe$
⟨3⟩3. $ll2InitialHistorySummary \in HistorySummaryType$
BY ⟨2⟩3

We then prove that, in the Memoir-Opt spec, the initial authenticator is a valid *MAC* for the initial history state binding. We will use the *MACComplete* property.

⟨3⟩4. $ValidateMAC($
$LL2NVRAM.symmetricKey$,
$ll2InitialHistoryStateBinding$,
$ll2InitialAuthenticator$)

In the Memoir-Opt spec, the initial authenticator is generated as a *MAC* of the initial history state binding.

⟨4⟩1. $ll2InitialAuthenticator =$
$GenerateMAC(LL2NVRAM.symmetricKey, ll2InitialHistoryStateBinding)$
⟨5⟩1. $LL2NVRAM.symmetricKey = symmetricKey$
BY ⟨2⟩2
⟨5⟩2. QED
BY ⟨5⟩1
DEF $ll2InitialAuthenticator$, $ll2InitialHistoryStateBinding$,
$ll2InitialHistorySummaryHash$, $initialStateHash$, $initialPrivateStateEnc$

We can thus use the *MACComplete* property to show that the generated *MAC* validates appropriately. To do this, we first need to prove some types.

⟨4⟩2. $LL2NVRAM.symmetricKey \in SymmetricKeyType$
BY ⟨2⟩1, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$
⟨4⟩3. $ll2InitialHistoryStateBinding \in HashType$
BY ⟨2⟩3

Then, we appeal to the *MACComplete* property in a straightforward way.

⟨4⟩4. QED
BY ⟨4⟩1, ⟨4⟩2, ⟨4⟩3, $MACComplete$

We then prove that, in the Memoir-Basic spec, the initial authenticator is generated as a *MAC* of the initial history state binding.

⟨3⟩5. $ll1InitialAuthenticator = GenerateMAC($
$LL2NVRAM.symmetricKey, ll1InitialHistoryStateBinding)$
⟨4⟩1. $LL2NVRAM.symmetricKey = symmetricKey$
BY ⟨2⟩2
⟨4⟩2. QED
BY ⟨4⟩1
DEF $ll1InitialAuthenticator$, $ll1InitialHistoryStateBinding$,
$initialStateHash$, $initialPrivateStateEnc$

The initial history summaries match across the two specs, as we proved above.

⟨3⟩6. $HistorySummariesMatch(BaseHashValue, ll2InitialHistorySummary, LL2NVRAM.hashBarrier)$
BY ⟨2⟩5

We then invoke the definition of the *AuthenticatorsMatch* predicate.

⟨3⟩7. QED
BY ⟨3⟩1, ⟨3⟩2, ⟨3⟩3, ⟨3⟩4, ⟨3⟩5, ⟨3⟩6

DEF $\mathit{AuthenticatorsMatch}$, $\mathit{ll2InitialAuthenticator}$,
$\mathit{ll1InitialHistoryStateBinding}$, $\mathit{ll2InitialHistoryStateBinding}$,
$\mathit{ll2InitialHistorySummaryHash}$, $\mathit{ll2InitialHistorySummary}$

The next six steps assert each conjunct in the $\mathit{LL1Init}$ predicate. For the $\mathit{LL1Disk}$ variable, we prove each field within the record separately.

⟨2⟩7. $\mathit{LL1Disk} = \mathit{ll1InitialUntrustedStorage}$

The $\mathit{LL1Disk}$'s public state equals the initial public state, because the refinement asserts a direct equality between this field for the two specs.

⟨3⟩1. $\mathit{LL1Disk.publicState} = \mathit{InitialPublicState}$
⟨4⟩1. $\mathit{LL2Disk.publicState} = \mathit{InitialPublicState}$
BY ⟨2⟩2
⟨4⟩2. $\mathit{LL1Disk.publicState} = \mathit{LL2Disk.publicState}$
BY ⟨2⟩1 DEF $\mathit{LL2Refinement}$
⟨4⟩3. QED
BY ⟨4⟩1, ⟨4⟩2

The $\mathit{LL1Disk}$'s private encrypted state equals the initial private encrypted state, because the refinement asserts a direct equality between this field for the two specs.

⟨3⟩2. $\mathit{LL1Disk.privateStateEnc} = \mathit{initialPrivateStateEnc}$
⟨4⟩1. $\mathit{LL2Disk.privateStateEnc} = \mathit{initialPrivateStateEnc}$
BY ⟨2⟩2 DEF $\mathit{initialPrivateStateEnc}$
⟨4⟩2. $\mathit{LL1Disk.privateStateEnc} = \mathit{LL2Disk.privateStateEnc}$
BY ⟨2⟩1 DEF $\mathit{LL2Refinement}$
⟨4⟩3. QED
BY ⟨4⟩1, ⟨4⟩2

The $\mathit{LL1Disk}$'s history summary equals the base hash value. There are four steps: (1) The refinement asserts that the corresponding fields from the two specs match according to the $\mathit{HistorySummariesMatch}$ predicate. (2) We prove that the initial history summary values match across the two specs. (3) We prove that the Memoir-Opt disk's history summary equals its initial history summary value. (4) We use the $\mathit{HistorySummariesMatchUniqueLemma}$ to prove that the field in the Memoir-Basic spec must equal the base hash value.

⟨3⟩3. $\mathit{LL1Disk.historySummary} = \mathit{BaseHashValue}$

The corresponding fields from the two specs match, thanks to the refinement.

⟨4⟩1. $\mathit{HistorySummariesMatch}($
$\mathit{LL1Disk.historySummary}$,
$\mathit{LL2Disk.historySummary}$,
$\mathit{LL2NVRAM.hashBarrier})$
BY ⟨2⟩1 DEF $\mathit{LL2Refinement}$

The initial history summaries match across the two specs, as we proved above.

⟨4⟩2. $\mathit{HistorySummariesMatch}(\mathit{BaseHashValue}, \mathit{ll2InitialHistorySummary}, \mathit{LL2NVRAM.hashBarrier})$
BY ⟨2⟩5

The history summary in the Memoir-Opt spec's disk equals the initial history summary, by the definition of $\mathit{LL2Init}$.

⟨4⟩3. $\mathit{LL2Disk.historySummary} = \mathit{ll2InitialHistorySummary}$
BY ⟨2⟩2 DEF $\mathit{ll2InitialHistorySummary}$

We use the $\mathit{HistorySummariesMatchUniqueLemma}$ to prove that the field in the Memoir-Basic spec equals the base hash value. This requires proving some types.

⟨4⟩4. QED
⟨5⟩1. $\mathit{LL1Disk.historySummary} \in \mathit{HashType}$
BY ⟨2⟩1 DEF $\mathit{LL2Refinement}$, $\mathit{LL1UntrustedStorageType}$
⟨5⟩2. $\mathit{BaseHashValue} \in \mathit{HashType}$
BY $\mathit{ConstantsTypeSafe}$
⟨5⟩3. $\mathit{LL2Disk.historySummary} \in \mathit{HistorySummaryType}$

BY ⟨2⟩1, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*

⟨5⟩4. $LL2NVRAM.hashBarrier \in HashType$

BY ⟨2⟩1, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*

⟨5⟩5. QED

BY ⟨4⟩1, ⟨4⟩2, ⟨4⟩3, ⟨5⟩1, ⟨5⟩2, ⟨5⟩3, ⟨5⟩4, *HistorySummariesMatchUniqueLemma*

The *LL1Disk*'s authenticator equals the initial authenticator. There are four steps: (1) The refinement asserts that the corresponding fields from the two specs match according to the *HistorySummariesMatch* predicate. (2) We prove that the initial authenticator values match across the two specs. (3) We prove that the Memoir-Opt disk's authenticator equals its initial authenticator value. (4) We use the *AuthenticatorsMatchUniqueLemma* to prove that the field in the Memoir-Basic spec must equal the initial authenticator value.

⟨3⟩4. $LL1Disk.authenticator = ll1InitialAuthenticator$

The corresponding fields from the two specs match, thanks to the refinement.

⟨4⟩1. $\exists\, symmetricKey1 \in SymmetricKeyType :$
$AuthenticatorsMatch($
$LL1Disk.authenticator,$
$LL2Disk.authenticator,$
$symmetricKey1,$
$LL2NVRAM.hashBarrier)$

BY ⟨2⟩1 DEF *LL2Refinement*

The corresponding initial authenticator values match across the two specs. We proved this above.

⟨4⟩2. $AuthenticatorsMatch($
$ll1InitialAuthenticator,$
$ll2InitialAuthenticator,$
$LL2NVRAM.symmetricKey,$
$LL2NVRAM.hashBarrier)$

BY ⟨2⟩6

The Memoir-Opt disk's authenticator equals its initial authenticator value, as asserted by the *LL2Init* predicate.

⟨4⟩3. $LL2Disk.authenticator = ll2InitialAuthenticator$

BY ⟨2⟩2

DEF *ll2InitialAuthenticator*, *ll2InitialHistoryStateBinding*,
ll2InitialHistorySummaryHash, *initialStateHash*, *initialPrivateStateEnc*

We use the *AuthenticatorsMatchUniqueLemma* to prove that the field in the Memoir-Basic spec equals the initial authenticator value. This requires proving some types.

⟨4⟩4. QED

⟨5⟩1. $LL1Disk.authenticator \in MACType$

BY ⟨2⟩1 DEF *LL2Refinement*, *LL1UntrustedStorageType*

⟨5⟩2. $ll1InitialAuthenticator \in MACType$

BY ⟨2⟩4

⟨5⟩3. $LL2Disk.authenticator \in MACType$

BY ⟨2⟩1, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*

⟨5⟩4. $LL2NVRAM.symmetricKey \in SymmetricKeyType$

BY ⟨2⟩1, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*

⟨5⟩5. $LL2NVRAM.hashBarrier \in HashType$

BY ⟨2⟩1, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*

⟨5⟩6. QED

BY ⟨4⟩1, ⟨4⟩2, ⟨4⟩3, ⟨5⟩1, ⟨5⟩2, ⟨5⟩3, ⟨5⟩4, ⟨5⟩5,
AuthenticatorsMatchUniqueLemma

The refinement asserts that the Disk record has the appropriate type.

⟨3⟩5. $LL1Disk \in LL1UntrustedStorageType$

BY ⟨2⟩1 DEF *LL2Refinement*

We use the *LL1DiskRecordCompositionLemma* to unify the field equalities into a record equality.

$\langle 3 \rangle 6$. QED
BY $\langle 3 \rangle 1$, $\langle 3 \rangle 2$, $\langle 3 \rangle 3$, $\langle 3 \rangle 4$, $\langle 3 \rangle 5$, *LL1DiskRecordCompositionLemma*
DEF *ll1InitialUntrustedStorage*

The second conjunct in the *LL1Init* predicate relates to the *LL1RAM* variable. We prove each field within the record separately.

$\langle 2 \rangle 8$. $LL1RAM = ll1InitialUntrustedStorage$

The *LL1RAM*'s public state equals the initial public state, because the refinement asserts a direct equality between this field for the two specs.

$\langle 3 \rangle 1$. $LL1RAM.publicState = InitialPublicState$
$\langle 4 \rangle 1$. $LL2RAM.publicState = InitialPublicState$
BY $\langle 2 \rangle 2$
$\langle 4 \rangle 2$. $LL1RAM.publicState = LL2RAM.publicState$
BY $\langle 2 \rangle 1$ DEF *LL2Refinement*
$\langle 4 \rangle 3$. QED
BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$

The *LL1RAM*'s private encrypted state equals the initial private encrypted state, because the refinement asserts a direct equality between this field for the two specs.

$\langle 3 \rangle 2$. $LL1RAM.privateStateEnc = initialPrivateStateEnc$
$\langle 4 \rangle 1$. $LL2RAM.privateStateEnc = initialPrivateStateEnc$
BY $\langle 2 \rangle 2$ DEF *initialPrivateStateEnc*
$\langle 4 \rangle 2$. $LL1RAM.privateStateEnc = LL2RAM.privateStateEnc$
BY $\langle 2 \rangle 1$ DEF *LL2Refinement*
$\langle 4 \rangle 3$. QED
BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$

The *LL1RAM*'s history summary equals the base hash value. There are four steps: (1) The refinement asserts that the corresponding fields from the two specs match according to the *HistorySummariesMatch* predicate. (2) We prove that the initial history summary values match across the two specs. (3) We prove that the Memoir-Opt RAM's history summary equals its initial history summary value. (4) We use the *HistorySummariesMatchUniqueLemma* to prove that the field in the Memoir-Basic spec must equal the base hash value.

$\langle 3 \rangle 3$. $LL1RAM.historySummary = BaseHashValue$

The corresponding fields from the two specs match, thanks to the refinement.

$\langle 4 \rangle 1$. $HistorySummariesMatch($
$LL1RAM.historySummary, LL2RAM.historySummary, LL2NVRAM.hashBarrier)$
BY $\langle 2 \rangle 1$ DEF *LL2Refinement*

The initial history summaries match across the two specs, as we proved above.

$\langle 4 \rangle 2$. $HistorySummariesMatch(BaseHashValue, ll2InitialHistorySummary, LL2NVRAM.hashBarrier)$
BY $\langle 2 \rangle 5$

The history summary in the Memoir-Opt spec's RAM equals the initial history summary, by the definition of *LL2Init*.

$\langle 4 \rangle 3$. $LL2RAM.historySummary = ll2InitialHistorySummary$
BY $\langle 2 \rangle 2$ DEF *ll2InitialHistorySummary*

We use the *HistorySummariesMatchUniqueLemma* to prove that the field in the Memoir-Basic spec equals the base hash value. This requires proving some types.

$\langle 4 \rangle 4$. QED
$\langle 5 \rangle 1$. $LL1RAM.historySummary \in HashType$
BY $\langle 2 \rangle 1$ DEF *LL2Refinement*, *LL1UntrustedStorageType*
$\langle 5 \rangle 2$. $BaseHashValue \in HashType$
BY *ConstantsTypeSafe*
$\langle 5 \rangle 3$. $LL2RAM.historySummary \in HistorySummaryType$
BY $\langle 2 \rangle 1$, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*
$\langle 5 \rangle 4$. $LL2NVRAM.hashBarrier \in HashType$

BY $\langle 2 \rangle 1$, *LL2SubtypeImplicationLemma* DEF *LL2SubtypeImplication*

$\langle 5 \rangle 5$. QED

BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$, $\langle 4 \rangle 3$, $\langle 5 \rangle 1$, $\langle 5 \rangle 2$, $\langle 5 \rangle 3$, $\langle 5 \rangle 4$, *HistorySummariesMatchUniqueLemma*

The *LL1RAM*'s authenticator equals the initial authenticator. There are four steps: (1) The refinement asserts that the corresponding fields from the two specs match according to the *HistorySummariesMatch* predicate. (2) We prove that the initial authenticator values match across the two specs. (3) We prove that the Memoir-Opt RAM's authenticator equals its initial authenticator value. (4) We use the *AuthenticatorsMatchUniqueLemma* to prove that the field in the Memoir-Basic spec must equal the initial authenticator value.

$\langle 3 \rangle 4$. $LL1RAM.authenticator = ll1InitialAuthenticator$

The corresponding fields from the two specs match, thanks to the refinement.

$\langle 4 \rangle 1$. $\exists\, symmetricKey1 \in SymmetricKeyType$:
$AuthenticatorsMatch($
$LL1RAM.authenticator,$
$LL2RAM.authenticator,$
$symmetricKey1,$
$LL2NVRAM.hashBarrier)$

BY $\langle 2 \rangle 1$ DEF *LL2Refinement*

The corresponding initial authenticator values match across the two specs. We proved this above.

$\langle 4 \rangle 2$. $AuthenticatorsMatch($
$ll1InitialAuthenticator,$
$ll2InitialAuthenticator,$
$LL2NVRAM.symmetricKey,$
$LL2NVRAM.hashBarrier)$

BY $\langle 2 \rangle 6$

The Memoir-Opt RAM's authenticator equals its initial authenticator value, as asserted by the *LL2Init* predicate.

$\langle 4 \rangle 3$. $LL2RAM.authenticator = ll2InitialAuthenticator$

BY $\langle 2 \rangle 2$

DEF *ll2InitialAuthenticator*, *ll2InitialHistoryStateBinding*,
ll2InitialHistorySummaryHash, *initialStateHash*, *initialPrivateStateEnc*

We use the *AuthenticatorsMatchUniqueLemma* to prove that the field in the Memoir-Basic spec equals the initial authenticator value. This requires proving some types.

$\langle 4 \rangle 4$. QED

$\langle 5 \rangle 1$. $LL1RAM.authenticator \in MACType$

BY $\langle 2 \rangle 1$ DEF *LL2Refinement*, *LL1UntrustedStorageType*

$\langle 5 \rangle 2$. $ll1InitialAuthenticator \in MACType$

BY $\langle 2 \rangle 4$

$\langle 5 \rangle 3$. $LL2RAM.authenticator \in MACType$

BY $\langle 2 \rangle 1$, *LL2SubtypeImplicationLemma* DEF *LL2SubtypeImplication*

$\langle 5 \rangle 4$. $LL2NVRAM.symmetricKey \in SymmetricKeyType$

BY $\langle 2 \rangle 1$, *LL2SubtypeImplicationLemma* DEF *LL2SubtypeImplication*

$\langle 5 \rangle 5$. $LL2NVRAM.hashBarrier \in HashType$

BY $\langle 2 \rangle 1$, *LL2SubtypeImplicationLemma* DEF *LL2SubtypeImplication*

$\langle 5 \rangle 6$. QED

BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$, $\langle 4 \rangle 3$, $\langle 5 \rangle 1$, $\langle 5 \rangle 2$, $\langle 5 \rangle 3$, $\langle 5 \rangle 4$, $\langle 5 \rangle 5$,
AuthenticatorsMatchUniqueLemma

The refinement asserts that the RAM record has the appropriate type.

$\langle 3 \rangle 5$. $LL1RAM \in LL1UntrustedStorageType$

BY $\langle 2 \rangle 1$ DEF *LL2Refinement*

We use the *LL1RAMRecordCompositionLemma* to unify the field equalities into a record equality.

$\langle 3 \rangle 6$. QED

BY $\langle 3 \rangle 1$, $\langle 3 \rangle 2$, $\langle 3 \rangle 3$, $\langle 3 \rangle 4$, $\langle 3 \rangle 5$, *LL1RAMRecordCompositionLemma*

DEF $ll1InitialUntrustedStorage$

The third conjunct in the $LL1Init$ predicate relates to the $LL1NVRAM$ variable.

$\langle 2 \rangle 9.$ $LL1NVRAM = ll1InitialTrustedStorage$

The $LL1NVRAM$'s history summary equals the base hash value. There are three steps: (1) The refinement asserts that the corresponding fields from the two specs match according to the $HistorySummariesMatch$ predicate. (2) We prove that the initial history summary values match across the two specs. (3) We prove that the Memoir-Opt $NVRAM$'s logical history summary equals its initial history summary value. (4) We use the $HistorySummariesMatchUniqueLemma$ to prove that the field in the Memoir-Basic spec must equal the base hash value.

$\langle 3 \rangle 1.$ $LL1NVRAM.historySummary = BaseHashValue$

The corresponding fields from the two specs match, thanks to the refinement.

$\langle 4 \rangle 1.$ $HistorySummariesMatch($
$LL1NVRAM.historySummary,\ LL2NVRAMLogicalHistorySummary,\ LL2NVRAM.hashBarrier)$
BY $\langle 2 \rangle 1$ DEF $LL2Refinement$

The initial history summaries match across the two specs, as we proved above.

$\langle 4 \rangle 2.$ $HistorySummariesMatch(BaseHashValue,\ ll2InitialHistorySummary,\ LL2NVRAM.hashBarrier)$
BY $\langle 2 \rangle 5$

The logical value of the history summary in the Memoir-Opt spec's $NVRAM$ equals the initial history summary. This follows from the definition of $LL2NVRAMLogicalHistorySummary$.

$\langle 4 \rangle 3.$ $LL2NVRAMLogicalHistorySummary = ll2InitialHistorySummary$
$\langle 5 \rangle 1.$ $LL2NVRAM.extensionInProgress = \text{FALSE}$
BY $\langle 2 \rangle 2$
$\langle 5 \rangle 2.$ $LL2NVRAM.historySummaryAnchor = BaseHashValue$
BY $\langle 2 \rangle 2$
$\langle 5 \rangle 3.$ $LL2SPCR = BaseHashValue$
BY $\langle 2 \rangle 2$
$\langle 5 \rangle 4.$ QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$, $\langle 5 \rangle 3$ DEF $LL2NVRAMLogicalHistorySummary,\ ll2InitialHistorySummary$

We use the $HistorySummariesMatchUniqueLemma$ to prove that the field in the Memoir-Basic spec equals the base hash value. This requires proving some types.

$\langle 4 \rangle 4.$ QED
$\langle 5 \rangle 1.$ $LL1NVRAM.historySummary \in HashType$
BY $\langle 2 \rangle 1$ DEF $LL2Refinement,\ LL1TrustedStorageType$
$\langle 5 \rangle 2.$ $BaseHashValue \in HashType$
BY $ConstantsTypeSafe$
$\langle 5 \rangle 3.$ $LL2NVRAMLogicalHistorySummary \in HistorySummaryType$
BY $\langle 2 \rangle 1$, $LL2NVRAMLogicalHistorySummaryTypeSafe$
$\langle 5 \rangle 4.$ $LL2NVRAM.hashBarrier \in HashType$
BY $\langle 2 \rangle 1$, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$
$\langle 5 \rangle 5.$ QED
BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$, $\langle 4 \rangle 3$, $\langle 5 \rangle 1$, $\langle 5 \rangle 2$, $\langle 5 \rangle 3$, $\langle 5 \rangle 4$, $HistorySummariesMatchUniqueLemma$

The symmetric key in the $LL1NVRAM$ matches its initial value by direct equality through the refinement.

$\langle 3 \rangle 2.$ $LL1NVRAM.symmetricKey = symmetricKey$
$\langle 4 \rangle 1.$ $LL1NVRAM.symmetricKey = LL2NVRAM.symmetricKey$
BY $\langle 2 \rangle 1$ DEF $LL2Refinement$
$\langle 4 \rangle 2.$ $LL2NVRAM.symmetricKey = symmetricKey$
BY $\langle 2 \rangle 2$
$\langle 4 \rangle 3.$ QED
BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$

The refinement asserts that the $NVRAM$ record has the appropriate type.

$\langle 3 \rangle 3.$ $LL1NVRAM \in LL1TrustedStorageType$

BY ⟨2⟩1 DEF *LL2Refinement*

We use the *LL1NVRAMRecordCompositionLemma* to unify the field equalities into a record equality.

⟨3⟩4. QED

BY ⟨3⟩1, ⟨3⟩2, ⟨3⟩3, *LL1NVRAMRecordCompositionLemma*
DEF *ll1InitialTrustedStorage*

The fourth conjunct in the *LL1Init* predicate relates to the *LL1AvailableInputs* variable. The proof is straightforward, because the equality is direct.

⟨2⟩10. $LL1AvailableInputs = InitialAvailableInputs$

⟨3⟩1. $LL2AvailableInputs = InitialAvailableInputs$

BY ⟨2⟩2

⟨3⟩2. $LL1AvailableInputs = LL2AvailableInputs$

BY ⟨2⟩1 DEF *LL2Refinement*

⟨3⟩3. QED

BY ⟨3⟩1, ⟨3⟩2

The fifth conjunct in the *LL1Init* predicate relates to the *LL1ObservedOutputs* variable. The proof is straightforward, because the equality is direct.

⟨2⟩11. $LL1ObservedOutputs = \{\}$

⟨3⟩1. $LL2ObservedOutputs = \{\}$

BY ⟨2⟩2

⟨3⟩2. $LL1ObservedOutputs = LL2ObservedOutputs$

BY ⟨2⟩1 DEF *LL2Refinement*

⟨3⟩3. QED

BY ⟨3⟩1, ⟨3⟩2

The sixth conjunct in the *LL1Init* predicate relates to the *LL1ObservedAuthenticators* variable. There are four steps: (1) The refinement asserts that the corresponding fields from the two specs match according to the *HistorySummariesMatch* predicate. (2) We prove that the initial authenticator values match across the two specs. (3) We prove that the Memoir-Opt set of observed authenticators equals its initial value. (4) We use the *AuthenticatorSetsMatchUniqueLemma* to prove that the variable in the Memoir-Basic spec must equal the initial value.

⟨2⟩12. $LL1ObservedAuthenticators = \{ll1InitialAuthenticator\}$

The corresponding variables from the two specs match, thanks to the refinement.

⟨3⟩1. *AuthenticatorSetsMatch*(
LL1ObservedAuthenticators,
LL2ObservedAuthenticators,
LL2NVRAM.symmetricKey,
LL2NVRAM.hashBarrier)

BY ⟨2⟩1 DEF *LL2Refinement*

The corresponding sets of initial authenticator values match across the two specs. Since the sets are singletons, this follows trivially from proving that one element in each set matches the other. And we proved that these elements match above.

⟨3⟩2. *AuthenticatorSetsMatch*(
{*ll1InitialAuthenticator*},
{*ll2InitialAuthenticator*},
LL2NVRAM.symmetricKey,
LL2NVRAM.hashBarrier)

⟨4⟩1. *AuthenticatorsMatch*(
ll1InitialAuthenticator,
ll2InitialAuthenticator,
LL2NVRAM.symmetricKey,
LL2NVRAM.hashBarrier)

BY ⟨2⟩6

⟨4⟩2. QED

BY $\langle 4 \rangle 1$ DEF *AuthenticatorSetsMatch*

The Memoir-Opt spec's set of observed authenticators equals its initial authenticator value, as asserted by the *LL2Init* predicate.

$\langle 3 \rangle 3$. $LL2ObservedAuthenticators = \{ll2InitialAuthenticator\}$
BY $\langle 2 \rangle 2$
DEF *ll2InitialAuthenticator*, *ll2InitialHistoryStateBinding*, *ll2InitialHistorySummaryHash*, *initialStateHash*, *initialPrivateStateEnc*

We use the *AuthenticatorSetsMatchUniqueLemma* to prove that the variable in the Memoir-Basic spec equals its initial value. This requires proving some types.

$\langle 3 \rangle 4$. QED
$\langle 4 \rangle 1$. $LL1ObservedAuthenticators \in$ SUBSET $MACType$
BY $\langle 2 \rangle 1$ DEF *LL2Refinement*
$\langle 4 \rangle 2$. $\{ll1InitialAuthenticator\} \in$ SUBSET $MACType$
$\langle 5 \rangle 1$. $ll1InitialAuthenticator \in MACType$
BY $\langle 2 \rangle 4$
$\langle 5 \rangle 2$. QED
BY $\langle 5 \rangle 1$
$\langle 4 \rangle 3$. $LL2ObservedAuthenticators \in$ SUBSET $MACType$
BY $\langle 2 \rangle 1$ DEF *LL2TypeInvariant*
$\langle 4 \rangle 4$. $LL2NVRAM.symmetricKey \in SymmetricKeyType$
BY $\langle 2 \rangle 1$, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*
$\langle 4 \rangle 5$. $LL2NVRAM.hashBarrier \in HashType$
BY $\langle 2 \rangle 1$, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*
$\langle 4 \rangle 6$. QED
BY $\langle 3 \rangle 1$, $\langle 3 \rangle 2$, $\langle 3 \rangle 3$, $\langle 4 \rangle 1$, $\langle 4 \rangle 2$, $\langle 4 \rangle 3$, $\langle 4 \rangle 4$, $\langle 4 \rangle 5$, *AuthenticatorSetsMatchUniqueLemma*

The conjunction of the above six conjuncts.

$\langle 2 \rangle 13$. QED
BY $\langle 2 \rangle 7$, $\langle 2 \rangle 8$, $\langle 2 \rangle 9$, $\langle 2 \rangle 10$, $\langle 2 \rangle 11$, $\langle 2 \rangle 12$
DEF *LL1Init*, *initialPrivateStateEnc*, *initialStateHash*, *ll1InitialHistoryStateBinding*, *ll1InitialAuthenticator*, *ll1InitialUntrustedStorage*, *ll1InitialTrustedStorage*

For the induction step, we will need the refinement and the type invariant to be true in both the unprimed and primed states.

$\langle 1 \rangle 3$. $(\land\ [LL2Next]_{LL2Vars}$
$\land\ LL2Refinement$
$\land\ LL2Refinement'$
$\land\ LL2TypeInvariant$
$\land\ LL2TypeInvariant')$
$\Rightarrow$
$[LL1Next]_{LL1Vars}$

We assume the antecedents.

$\langle 2 \rangle 1$. HAVE $\land\ [LL2Next]_{LL2Vars}$
$\land\ LL2Refinement$
$\land\ LL2Refinement'$
$\land\ LL2TypeInvariant$
$\land\ LL2TypeInvariant'$

We then prove that each step in the Memoir-Opt spec refines to a step in the Memoir-Basic spec. First, a Memoir-Opt stuttering step refines to a Memoir-Basic stuttering step. We prove the UNCHANGED status for each Memoir-Basic variable in turn, using the lemmas we have proven for this purpose.

$\langle 2 \rangle 2$. UNCHANGED $LL2Vars \Rightarrow$ UNCHANGED $LL1Vars$
$\langle 3 \rangle 1$. HAVE UNCHANGED $LL2Vars$

$\langle 3 \rangle 2.$ UNCHANGED $LL1AvailableInputs$
BY $\langle 2 \rangle 1$, $\langle 3 \rangle 1$, $UnchangedAvailableInputsLemma$ DEF $LL2Vars$
$\langle 3 \rangle 3.$ UNCHANGED $LL1ObservedOutputs$
BY $\langle 2 \rangle 1$, $\langle 3 \rangle 1$, $UnchangedObservedOutputsLemma$ DEF $LL2Vars$
$\langle 3 \rangle 4.$ UNCHANGED $LL1ObservedAuthenticators$
BY $\langle 2 \rangle 1$, $\langle 3 \rangle 1$, $UnchangedObservedAuthenticatorsLemma$ DEF $LL2Vars$
$\langle 3 \rangle 5.$ UNCHANGED $LL1Disk$
BY $\langle 2 \rangle 1$, $\langle 3 \rangle 1$, $UnchangedDiskLemma$ DEF $LL2Vars$
$\langle 3 \rangle 6.$ UNCHANGED $LL1RAM$
BY $\langle 2 \rangle 1$, $\langle 3 \rangle 1$, $UnchangedRAMLemma$ DEF $LL2Vars$
$\langle 3 \rangle 7.$ UNCHANGED $LL1NVRAM$
BY $\langle 2 \rangle 1$, $\langle 3 \rangle 1$, $UnchangedNVRAMLemma$ DEF $LL2Vars$
$\langle 3 \rangle 8.$ QED
BY $\langle 3 \rangle 2$, $\langle 3 \rangle 3$, $\langle 3 \rangle 4$, $\langle 3 \rangle 5$, $\langle 3 \rangle 6$, $\langle 3 \rangle 7$ DEF $LL1Vars$

A Memoir-Opt *LL2MakeInputAvailable* action refines to a Memoir-Basic *LL1MakeInputAvailable* action.

$\langle 2 \rangle 3.$ $LL2MakeInputAvailable \Rightarrow LL1MakeInputAvailable$
$\langle 3 \rangle 1.$ HAVE $LL2MakeInputAvailable$
$\langle 3 \rangle 2.$ PICK $input \in InputType : LL2MakeInputAvailable!(input)$
BY $\langle 3 \rangle 1$ DEF $LL2MakeInputAvailable$

We prove each conjunct in the *LL1MakeInputAvailable* action separately.

$\langle 3 \rangle 3.$ $input \notin LL1AvailableInputs$
$\langle 4 \rangle 1.$ $input \notin LL2AvailableInputs$
BY $\langle 3 \rangle 2$
$\langle 4 \rangle 2.$ $LL1AvailableInputs = LL2AvailableInputs$
BY $\langle 2 \rangle 1$ DEF $LL2Refinement$
$\langle 4 \rangle 3.$ $LL1AvailableInputs' = LL2AvailableInputs'$
BY $\langle 2 \rangle 1$ DEF $LL2Refinement$
$\langle 4 \rangle 4.$ QED
BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$, $\langle 4 \rangle 3$
$\langle 3 \rangle 4.$ $LL1AvailableInputs' = LL1AvailableInputs \cup \{input\}$
$\langle 4 \rangle 1.$ $LL2AvailableInputs' = LL2AvailableInputs \cup \{input\}$
BY $\langle 3 \rangle 2$
$\langle 4 \rangle 2.$ $LL1AvailableInputs = LL2AvailableInputs$
BY $\langle 2 \rangle 1$ DEF $LL2Refinement$
$\langle 4 \rangle 3.$ $LL1AvailableInputs' = LL2AvailableInputs'$
BY $\langle 2 \rangle 1$ DEF $LL2Refinement$
$\langle 4 \rangle 4.$ QED
BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$, $\langle 4 \rangle 3$
$\langle 3 \rangle 5.$ UNCHANGED $LL1Disk$
BY $\langle 2 \rangle 1$, $\langle 3 \rangle 2$, $UnchangedDiskLemma$
$\langle 3 \rangle 6.$ UNCHANGED $LL1RAM$
BY $\langle 2 \rangle 1$, $\langle 3 \rangle 2$, $UnchangedRAMLemma$
$\langle 3 \rangle 7.$ UNCHANGED $LL1NVRAM$
BY $\langle 2 \rangle 1$, $\langle 3 \rangle 2$, $UnchangedNVRAMLemma$
$\langle 3 \rangle 8.$ UNCHANGED $LL1ObservedOutputs$
BY $\langle 2 \rangle 1$, $\langle 3 \rangle 2$, $UnchangedObservedOutputsLemma$
$\langle 3 \rangle 9.$ UNCHANGED $LL1ObservedAuthenticators$
BY $\langle 2 \rangle 1$, $\langle 3 \rangle 2$, $UnchangedObservedAuthenticatorsLemma$
$\langle 3 \rangle$.QED
BY $\langle 3 \rangle 3$, $\langle 3 \rangle 4$, $\langle 3 \rangle 5$, $\langle 3 \rangle 6$, $\langle 3 \rangle 7$, $\langle 3 \rangle 8$, $\langle 3 \rangle 9$ DEF $LL1MakeInputAvailable$

A Memoir-Opt *LL2PerformOperation* action refines to a Memoir-Basic *LL1PerformOperation* action.

$\langle 2 \rangle 4.$ $LL2PerformOperation \Rightarrow LL1PerformOperation$

We assume the antecedent.

$\langle 3 \rangle 1.$ HAVE $LL2PerformOperation$

We pick an input that satisfies the *LL2PerformOperation* predicate.

$\langle 3 \rangle 2.$ PICK $input \in LL2AvailableInputs : LL2PerformOperation!(input)!1$
BY $\langle 3 \rangle 1$ DEF $LL2PerformOperation$

We re-state the definitions from *LL2PerformOperation*.

$\langle 3 \rangle$ $ll2HistorySummaryHash \triangleq$
$Hash(LL2RAM.historySummary.anchor, LL2RAM.historySummary.extension)$
$\langle 3 \rangle$ $ll2StateHash \triangleq Hash(LL2RAM.publicState, LL2RAM.privateStateEnc)$
$\langle 3 \rangle$ $ll2HistoryStateBinding \triangleq Hash(ll2HistorySummaryHash, ll2StateHash)$
$\langle 3 \rangle$ $ll2PrivateState \triangleq SymmetricDecrypt(LL2NVRAM.symmetricKey, LL2RAM.privateStateEnc)$
$\langle 3 \rangle$ $ll2SResult \triangleq Service(LL2RAM.publicState, ll2PrivateState, input)$
$\langle 3 \rangle$ $ll2NewPrivateStateEnc \triangleq$
$SymmetricEncrypt(LL2NVRAM.symmetricKey, ll2SResult.newPrivateState)$
$\langle 3 \rangle$ $ll2CurrentHistorySummary \triangleq [$
$anchor \mapsto LL2NVRAM.historySummaryAnchor,$
$extension \mapsto LL2SPCR]$
$\langle 3 \rangle$ $ll2NewHistorySummary \triangleq$
$Successor(ll2CurrentHistorySummary, input, LL2NVRAM.hashBarrier)$
$\langle 3 \rangle$ $ll2NewHistorySummaryHash \triangleq$
$Hash(ll2NewHistorySummary.anchor, ll2NewHistorySummary.extension)$
$\langle 3 \rangle$ $ll2NewStateHash \triangleq Hash(ll2SResult.newPublicState, ll2NewPrivateStateEnc)$
$\langle 3 \rangle$ $ll2NewHistoryStateBinding \triangleq Hash(ll2NewHistorySummaryHash, ll2NewStateHash)$
$\langle 3 \rangle$ $ll2NewAuthenticator \triangleq GenerateMAC(LL2NVRAM.symmetricKey, ll2NewHistoryStateBinding)$

We prove that the definitions from *LL2PerformOperation* satisfy their types, using the *LL2PerformOperationDefsTypeSafeLemma*.

$\langle 3 \rangle 3.$ $\land$ $ll2HistorySummaryHash \in HashType$
$\land$ $ll2StateHash \in HashType$
$\land$ $ll2HistoryStateBinding \in HashType$
$\land$ $ll2PrivateState \in PrivateStateType$
$\land$ $ll2SResult \in ServiceResultType$
$\land$ $ll2SResult.newPublicState \in PublicStateType$
$\land$ $ll2SResult.newPrivateState \in PrivateStateType$
$\land$ $ll2SResult.output \in OutputType$
$\land$ $ll2NewPrivateStateEnc \in PrivateStateEncType$
$\land$ $ll2CurrentHistorySummary \in HistorySummaryType$
$\land$ $ll2CurrentHistorySummary.anchor \in HashType$
$\land$ $ll2CurrentHistorySummary.extension \in HashType$
$\land$ $ll2NewHistorySummary \in HistorySummaryType$
$\land$ $ll2NewHistorySummary.anchor \in HashType$
$\land$ $ll2NewHistorySummary.extension \in HashType$
$\land$ $ll2NewHistorySummaryHash \in HashType$
$\land$ $ll2NewStateHash \in HashType$
$\land$ $ll2NewHistoryStateBinding \in HashType$
$\land$ $ll2NewAuthenticator \in MACType$

$\langle 4 \rangle 1.$ $input \in LL2AvailableInputs$
BY $\langle 3 \rangle 2$

$\langle 4 \rangle 2.$ $LL2TypeInvariant$
BY $\langle 2 \rangle 1$

$\langle 4 \rangle 3.$ QED

BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$, $LL2PerformOperationDefsTypeSafeLemma$

We hide the definitions.

$\langle 3 \rangle$ HIDE DEF $ll2HistorySummaryHash$, $ll2StateHash$, $ll2HistoryStateBinding$, $ll2PrivateState$, $ll2SResult$, $ll2NewPrivateStateEnc$, $ll2CurrentHistorySummary$, $ll2NewHistorySummary$, $ll2NewHistorySummaryHash$, $ll2NewStateHash$, $ll2NewHistoryStateBinding$, $ll2NewAuthenticator$

One fact that will be needed many places is that the input is in the Memoir-Basic set of available inputs.

$\langle 3 \rangle 4.$ $input \in LL1AvailableInputs$
  $\langle 4 \rangle 1.$ $input \in LL2AvailableInputs$
    BY $\langle 3 \rangle 2$
  $\langle 4 \rangle 2.$ $LL1AvailableInputs = LL2AvailableInputs$
    BY $\langle 2 \rangle 1$ DEF $LL2Refinement$
  $\langle 4 \rangle 3.$ QED
    BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$

We re-state the definitions from $LL1PerformOperation$.

$\langle 3 \rangle$ $ll1StateHash \triangleq Hash(LL1RAM.publicState, LL1RAM.privateStateEnc)$
$\langle 3 \rangle$ $ll1HistoryStateBinding \triangleq Hash(LL1RAM.historySummary, ll1StateHash)$
$\langle 3 \rangle$ $ll1PrivateState \triangleq SymmetricDecrypt(LL1NVRAM.symmetricKey, LL1RAM.privateStateEnc)$
$\langle 3 \rangle$ $ll1sResult \triangleq Service(LL1RAM.publicState, ll1PrivateState, input)$
$\langle 3 \rangle$ $ll1NewPrivateStateEnc \triangleq$
    $SymmetricEncrypt(LL1NVRAM.symmetricKey, ll1sResult.newPrivateState)$
$\langle 3 \rangle$ $ll1NewHistorySummary \triangleq Hash(LL1NVRAM.historySummary, input)$
$\langle 3 \rangle$ $ll1NewStateHash \triangleq Hash(ll1sResult.newPublicState, ll1NewPrivateStateEnc)$
$\langle 3 \rangle$ $ll1NewHistoryStateBinding \triangleq Hash(ll1NewHistorySummary, ll1NewStateHash)$
$\langle 3 \rangle$ $ll1NewAuthenticator \triangleq GenerateMAC(LL1NVRAM.symmetricKey, ll1NewHistoryStateBinding)$

We prove that the definitions from $LL1PerformOperation$ satisfy their types, using the $LL1PerformOperationDefsTypeSafeLemma$ and the $TypeSafetyRefinementLemma$.

$\langle 3 \rangle 5.$ $\land$ $ll1StateHash \in HashType$
    $\land$ $ll1HistoryStateBinding \in HashType$
    $\land$ $ll1PrivateState \in PrivateStateType$
    $\land$ $ll1sResult \in ServiceResultType$
    $\land$ $ll1sResult.newPublicState \in PublicStateType$
    $\land$ $ll1sResult.newPrivateState \in PrivateStateType$
    $\land$ $ll1sResult.output \in OutputType$
    $\land$ $ll1NewPrivateStateEnc \in PrivateStateEncType$
    $\land$ $ll1NewHistorySummary \in HashType$
    $\land$ $ll1NewStateHash \in HashType$
    $\land$ $ll1NewHistoryStateBinding \in HashType$
    $\land$ $ll1NewAuthenticator \in MACType$
  $\langle 4 \rangle 1.$ $input \in LL1AvailableInputs$
    BY $\langle 3 \rangle 4$
  $\langle 4 \rangle 2.$ $LL1TypeInvariant$
    BY $\langle 2 \rangle 1$, $TypeSafetyRefinementLemma$
  $\langle 4 \rangle 3.$ QED
    BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$, $LL1PerformOperationDefsTypeSafeLemma$

We hide the definitions from $LL1PerformOperation$.

$\langle 3 \rangle$ HIDE DEF $ll1StateHash$, $ll1HistoryStateBinding$, $ll1PrivateState$, $ll1sResult$, $ll1NewPrivateStateEnc$, $ll1NewHistorySummary$, $ll1NewStateHash$, $ll1NewHistoryStateBinding$, $ll1NewAuthenticator$

We prove the correspondences between the definitions in $LL1PerformOperation$ and $LL2PerformOperation$. The state hashes are directly equal.

$\langle 3 \rangle 6.$ $ll1StateHash = ll2StateHash$
  $\langle 4 \rangle 1.$ $LL1RAM.publicState = LL2RAM.publicState$

BY $\langle 2 \rangle 1$ DEF *LL2Refinement*

$\langle 4 \rangle 2.$ $LL1RAM.privateStateEnc = LL2RAM.privateStateEnc$

BY $\langle 2 \rangle 1$ DEF *LL2Refinement*

$\langle 4 \rangle 3.$ QED

BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$ DEF *ll1StateHash*, *ll2StateHash*

The private states are directly equal across the two specs.

$\langle 3 \rangle 7.$ $ll1PrivateState = ll2PrivateState$

$\langle 4 \rangle 1.$ $LL1NVRAM.symmetricKey = LL2NVRAM.symmetricKey$

BY $\langle 2 \rangle 1$ DEF *LL2Refinement*

$\langle 4 \rangle 2.$ $LL1RAM.privateStateEnc = LL2RAM.privateStateEnc$

BY $\langle 2 \rangle 1$ DEF *LL2Refinement*

$\langle 4 \rangle 3.$ QED

BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$ DEF *ll1PrivateState*, *ll2PrivateState*

The service results are directly equal across the two specs.

$\langle 3 \rangle 8.$ $ll1sResult = ll2SResult$

$\langle 4 \rangle 1.$ $LL1RAM.publicState = LL2RAM.publicState$

BY $\langle 2 \rangle 1$ DEF *LL2Refinement*

$\langle 4 \rangle 2.$ $ll1PrivateState = ll2PrivateState$

BY $\langle 3 \rangle 7$

$\langle 4 \rangle 3.$ QED

BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$ DEF *ll1sResult*, *ll2SResult*

The new encrypted private states are directly equal across the two specs.

$\langle 3 \rangle 9.$ $ll1NewPrivateStateEnc = ll2NewPrivateStateEnc$

$\langle 4 \rangle 1.$ $LL1NVRAM.symmetricKey = LL2NVRAM.symmetricKey$

BY $\langle 2 \rangle 1$ DEF *LL2Refinement*

$\langle 4 \rangle 2.$ $ll1sResult.newPrivateState = ll2SResult.newPrivateState$

BY $\langle 3 \rangle 8$

$\langle 4 \rangle 3.$ QED

BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$ DEF *ll1NewPrivateStateEnc*, *ll2NewPrivateStateEnc*

The new history summaries match across the two specs, as specified by the *HistorySummariesMatch* predicate.

$\langle 3 \rangle 10.$ $HistorySummariesMatch($
$\quad ll1NewHistorySummary, ll2NewHistorySummary, LL2NVRAM.hashBarrier')$

First, we prove that the *HistorySummariesMatch* predicate equals the *HistorySummariesMatchRecursion* predicate in this case. We assert each condition required by the definition of the predicate.

$\langle 4 \rangle 1.$ $HistorySummariesMatch($
$\quad ll1NewHistorySummary, ll2NewHistorySummary, LL2NVRAM.hashBarrier') =$
$HistorySummariesMatchRecursion($
$\quad ll1NewHistorySummary, ll2NewHistorySummary, LL2NVRAM.hashBarrier')$

We begin by proving the types for the *HistorySummariesMatchRecursion* predicate.

$\langle 5 \rangle 1.$ $ll1NewHistorySummary \in HashType$

BY $\langle 3 \rangle 5$

$\langle 5 \rangle 2.$ $ll2NewHistorySummary \in HistorySummaryType$

BY $\langle 3 \rangle 3$

$\langle 5 \rangle 3.$ $LL2NVRAM.hashBarrier' \in HashType$

BY $\langle 2 \rangle 1$, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*

$\langle 5 \rangle$ $ll2InitialHistorySummary \triangleq [anchor \mapsto BaseHashValue, extension \mapsto BaseHashValue]$

We then prove that this is not the base case for the *HistorySummariesMatch* predicate.

$\langle 5 \rangle 4.$ $ll2NewHistorySummary \neq ll2InitialHistorySummary$

This proof has a lot of sub-steps, but it is pretty simple. We just use the *BaseHashValueUnique* property to show that the extension field in the *ll2NewHistorySummary* record cannot match the base hash value, which is the value of the extension field in the initial history summary.

$\langle 6 \rangle 1.$ *ll2NewHistorySummary* =
Successor(*ll2CurrentHistorySummary*, *input*, *LL2NVRAM.hashBarrier*)
BY DEF *ll2NewHistorySummary*
$\langle 6 \rangle 2.$ *ll2NewHistorySummary.extension* =
Hash(*ll2CurrentHistorySummary.extension*, *Hash*(*LL2NVRAM.hashBarrier*, *input*))
BY $\langle 6 \rangle 1$ DEF *Successor*
$\langle 6 \rangle 3.$ *ll2NewHistorySummary.extension* $\neq$ *BaseHashValue*
$\langle 7 \rangle 1.$ *ll2CurrentHistorySummary.extension* $\in$ *HashDomain*
$\langle 8 \rangle 1.$ *ll2CurrentHistorySummary.extension* $\in$ *HashType*
$\langle 9 \rangle 1.$ *ll2CurrentHistorySummary.extension* = *LL2SPCR*
BY DEF *ll2CurrentHistorySummary*
$\langle 9 \rangle 2.$ *LL2SPCR* $\in$ *HashType*
BY $\langle 2 \rangle 1$ DEF *LL2TypeInvariant*
$\langle 9 \rangle 3.$ QED
BY $\langle 9 \rangle 1$, $\langle 9 \rangle 2$
$\langle 8 \rangle 2.$ QED
BY $\langle 8 \rangle 1$ DEF *HashDomain*
$\langle 7 \rangle 2.$ *Hash*(*LL2NVRAM.hashBarrier*, *input*) $\in$ *HashDomain*
$\langle 8 \rangle 1.$ *Hash*(*LL2NVRAM.hashBarrier*, *input*) $\in$ *HashType*
$\langle 9 \rangle 1.$ *LL2NVRAM.hashBarrier* $\in$ *HashDomain*
$\langle 10 \rangle 1.$ *LL2NVRAM.hashBarrier* $\in$ *HashType*
BY $\langle 2 \rangle 1$, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*
$\langle 10 \rangle 2.$ QED
BY $\langle 10 \rangle 1$ DEF *HashDomain*
$\langle 9 \rangle 2.$ *input* $\in$ *HashDomain*
$\langle 10 \rangle 1.$ *input* $\in$ *InputType*
$\langle 11 \rangle 1.$ *input* $\in$ *LL2AvailableInputs*
BY $\langle 3 \rangle 2$
$\langle 11 \rangle 2.$ *LL2AvailableInputs* $\subseteq$ *InputType*
BY $\langle 2 \rangle 1$ DEF *LL2TypeInvariant*
$\langle 11 \rangle 3.$ QED
BY $\langle 11 \rangle 1$, $\langle 11 \rangle 2$
$\langle 10 \rangle 2.$ QED
BY $\langle 10 \rangle 1$ DEF *HashDomain*
$\langle 9 \rangle 3.$ QED
BY $\langle 9 \rangle 1$, $\langle 9 \rangle 2$, *HashTypeSafe*
$\langle 8 \rangle 2.$ QED
BY $\langle 8 \rangle 1$ DEF *HashDomain*
$\langle 7 \rangle 3.$ QED
BY $\langle 6 \rangle 2$, $\langle 7 \rangle 1$, $\langle 7 \rangle 2$, *BaseHashValueUnique*
$\langle 6 \rangle 4.$ QED
BY $\langle 6 \rangle 3$

Since this is not the base case, the *HistorySummariesMatch* predicate equals the *HistorySummariesMatchRecursion* predicate.

$\langle 5 \rangle 5.$ QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$, $\langle 5 \rangle 3$, $\langle 5 \rangle 4$, *HistorySummariesMatchDefinition*

Then, we prove that the *HistorySummariesMatchRecursion* predicate is satisfied. We assert each condition required by the definition of the predicate.

$\langle 4 \rangle 2.$ *HistorySummariesMatchRecursion*(
ll1NewHistorySummary, *ll2NewHistorySummary*, *LL2NVRAM.hashBarrier*$'$)

We begin by proving the types for the existentially quantified variables in the *HistorySummariesMatchRecursion* predicate.

$\langle 5\rangle 1.$ $input \in InputType$
  $\langle 6\rangle 1.$ $input \in LL2AvailableInputs$
    BY $\langle 3\rangle 2$
  $\langle 6\rangle 2.$ $LL2AvailableInputs \subseteq InputType$
    BY $\langle 2\rangle 1$ DEF $LL2TypeInvariant$
  $\langle 6\rangle 3.$ QED
    BY $\langle 6\rangle 1$, $\langle 6\rangle 2$
$\langle 5\rangle 2.$ $LL1NVRAM.historySummary \in HashType$
  BY $\langle 2\rangle 1$ DEF $LL2Refinement$, $LL1TrustedStorageType$
$\langle 5\rangle 3.$ $LL2NVRAMLogicalHistorySummary \in HistorySummaryType$
  BY $\langle 2\rangle 1$, $LL2NVRAMLogicalHistorySummaryTypeSafe$

We then prove the three conjuncts in the *HistorySummariesMatchRecursion* predicate. The first conjunct follows directly from the refinement.

$\langle 5\rangle 4.$ $HistorySummariesMatch($
      $LL1NVRAM.historySummary,$
      $LL2NVRAMLogicalHistorySummary,$
      $LL2NVRAM.hashBarrier')$
  $\langle 6\rangle 1.$ UNCHANGED $LL2NVRAM.hashBarrier$
    BY $\langle 3\rangle 2$
  $\langle 6\rangle 2.$ QED
    BY $\langle 2\rangle 1$, $\langle 6\rangle 1$ DEF $LL2Refinement$

The second conjunct in the *HistorySummariesMatchRecursion* predicate is true by definition.

$\langle 5\rangle 5.$ $ll1NewHistorySummary = Hash(LL1NVRAM.historySummary, input)$
  BY DEF $ll1NewHistorySummary$

The third conjunct in the *HistorySummariesMatchRecursion* predicate is true because the *ll2NewHistorySummary* definition in the *LL2PerformOperation* action yields a value that is the successor of the logical history summary in the *NVRAM*.

$\langle 5\rangle 6.$ $LL2HistorySummaryIsSuccessor($
      $ll2NewHistorySummary,$
      $LL2NVRAMLogicalHistorySummary,$
      $input,$
      $LL2NVRAM.hashBarrier')$
  $\langle 6\rangle 1.$ $LL2NVRAMLogicalHistorySummary = ll2CurrentHistorySummary$
    $\langle 7\rangle 1.$ CASE $LL2NVRAM.extensionInProgress = \text{TRUE}$
      $\langle 8\rangle 2.$ $LL2NVRAMLogicalHistorySummary = [$
          $anchor \mapsto LL2NVRAM.historySummaryAnchor,$
          $extension \mapsto LL2SPCR]$
        $\langle 9\rangle 1.$ $LL2SPCR \neq BaseHashValue$
          BY $\langle 3\rangle 2$, $\langle 7\rangle 1$
        $\langle 9\rangle 2.$ QED
          BY $\langle 7\rangle 1$, $\langle 9\rangle 1$ DEF $LL2NVRAMLogicalHistorySummary$
      $\langle 8\rangle 3.$ $ll2CurrentHistorySummary = [$
          $anchor \mapsto LL2NVRAM.historySummaryAnchor,$
          $extension \mapsto LL2SPCR]$
        BY DEF $ll2CurrentHistorySummary$
      $\langle 8\rangle 4.$ QED
        BY $\langle 8\rangle 2$, $\langle 8\rangle 3$
    $\langle 7\rangle 2.$ CASE $LL2NVRAM.extensionInProgress = \text{FALSE}$
      $\langle 8\rangle 1.$ $LL2NVRAMLogicalHistorySummary = [$
          $anchor \mapsto LL2NVRAM.historySummaryAnchor,$
          $extension \mapsto BaseHashValue]$
        BY $\langle 7\rangle 2$ DEF $LL2NVRAMLogicalHistorySummary$

$\langle 8\rangle 2.$ $ll2CurrentHistorySummary = [$
$anchor \mapsto LL2NVRAM.historySummaryAnchor,$
$extension \mapsto LL2SPCR]$
BY DEF $ll2CurrentHistorySummary$
$\langle 8\rangle 3.$ $LL2SPCR = BaseHashValue$
BY $\langle 3\rangle 2$, $\langle 7\rangle 2$
$\langle 8\rangle 4.$ QED
BY $\langle 8\rangle 1$, $\langle 8\rangle 2$, $\langle 8\rangle 3$
$\langle 7\rangle 3.$ $LL2NVRAM.extensionInProgress \in$ BOOLEAN
BY $\langle 2\rangle 1$, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$
$\langle 7\rangle 4.$ QED
BY $\langle 7\rangle 1$, $\langle 7\rangle 2$, $\langle 7\rangle 3$
$\langle 6\rangle 2.$ $ll2NewHistorySummary =$
$Successor(LL2NVRAMLogicalHistorySummary, input, LL2NVRAM.hashBarrier)$
BY $\langle 6\rangle 1$ DEF $ll2NewHistorySummary$
$\langle 6\rangle 3.$ UNCHANGED $LL2NVRAM.hashBarrier$
BY $\langle 3\rangle 2$
$\langle 6\rangle 4.$ QED
BY $\langle 6\rangle 2$, $\langle 6\rangle 3$ DEF $LL2HistorySummaryIsSuccessor$

All conjuncts in the *HistorySummariesMatchRecursion* predicate are satisfied.

$\langle 5\rangle 7.$ QED
BY $\langle 5\rangle 1$, $\langle 5\rangle 2$, $\langle 5\rangle 3$, $\langle 5\rangle 4$, $\langle 5\rangle 5$, $\langle 5\rangle 6$ DEF $HistorySummariesMatchRecursion$

Since the *HistorySummariesMatch* predicate equals the *HistorySummariesMatchRecursion* predicate, and the latter predicate is satisfied, the former predicate is satisfied.

$\langle 4\rangle 3.$ QED
BY $\langle 4\rangle 1$, $\langle 4\rangle 2$

The new state hashes are directly equal across the two specs.

$\langle 3\rangle 11.$ $ll1NewStateHash = ll2NewStateHash$
$\langle 4\rangle 1.$ $ll1sResult.newPublicState = ll2SResult.newPublicState$
BY $\langle 3\rangle 8$
$\langle 4\rangle 2.$ $ll1NewPrivateStateEnc = ll2NewPrivateStateEnc$
BY $\langle 3\rangle 9$
$\langle 4\rangle 3.$ QED
BY $\langle 4\rangle 1$, $\langle 4\rangle 2$ DEF $ll1NewStateHash$, $ll2NewStateHash$

The new authenticators match across the two specs, as specified by the *AuthenticatorsMatch* predicate.

$\langle 3\rangle 12.$ $AuthenticatorsMatch($
$ll1NewAuthenticator,$
$ll2NewAuthenticator,$
$LL2NVRAM.symmetricKey',$
$LL2NVRAM.hashBarrier')$

First, we prove some types needed by the definition of the *AuthenticatorsMatch* predicate.

$\langle 4\rangle 1.$ $ll2NewStateHash \in HashType$
BY $\langle 3\rangle 3$
$\langle 4\rangle 2.$ $ll1NewHistorySummary \in HashType$
BY $\langle 3\rangle 5$
$\langle 4\rangle 3.$ $ll2NewHistorySummary \in HistorySummaryType$
BY $\langle 3\rangle 3$

We then prove that, in the Memoir-Opt spec, the new authenticator is a valid *MAC* for the new history state binding. We will use the *MACComplete* property.

$\langle 4\rangle 4.$ $ValidateMAC(LL2NVRAM.symmetricKey', ll2NewHistoryStateBinding, ll2NewAuthenticator)$

In the Memoir-Opt spec, the new authenticator is generated as a *MAC* of the new history state binding.

$\langle 5\rangle 1.\ ll2NewAuthenticator =$
$\quad GenerateMAC(LL2NVRAM.symmetricKey', ll2NewHistoryStateBinding)$

$\langle 6\rangle 1.$ UNCHANGED $LL2NVRAM.symmetricKey$
BY $\langle 3\rangle 2$

$\langle 6\rangle 2.$ QED
BY $\langle 6\rangle 1$ DEF $ll2NewAuthenticator$

We can thus use the *MACComplete* property to show that the generated *MAC* validates appropriately. To do this, we first need to prove some types.

$\langle 5\rangle 2.\ LL2NVRAM.symmetricKey' \in SymmetricKeyType$
BY $\langle 2\rangle 1$, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*

$\langle 5\rangle 3.\ ll2NewHistoryStateBinding \in HashType$
BY $\langle 3\rangle 3$

Then, we appeal to the *MACComplete* property in a straightforward way.

$\langle 5\rangle 4.$ QED
BY $\langle 5\rangle 1$, $\langle 5\rangle 2$, $\langle 5\rangle 3$, *MACComplete*

We then prove that, in the Memoir-Basic spec, the new authenticator is generated as a *MAC* of the new history state binding.

$\langle 4\rangle 5.\ ll1NewAuthenticator = GenerateMAC(LL2NVRAM.symmetricKey', ll1NewHistoryStateBinding)$

$\langle 5\rangle 1.\ LL1NVRAM.symmetricKey = LL2NVRAM.symmetricKey'$

$\langle 6\rangle 1.\ LL1NVRAM.symmetricKey = LL2NVRAM.symmetricKey$
BY $\langle 2\rangle 1$ DEF *LL2Refinement*

$\langle 6\rangle 2.$ UNCHANGED $LL2NVRAM.symmetricKey$
BY $\langle 3\rangle 2$

$\langle 6\rangle 3.$ QED
BY $\langle 6\rangle 1$, $\langle 6\rangle 2$

$\langle 5\rangle 2.$ QED
BY $\langle 5\rangle 1$ DEF $ll1NewAuthenticator$

The new history summaries match across the two specs, as we proved above.

$\langle 4\rangle 6.\ HistorySummariesMatch($
$\quad ll1NewHistorySummary, ll2NewHistorySummary, LL2NVRAM.hashBarrier')$
BY $\langle 3\rangle 10$

We then invoke the definition of the *AuthenticatorsMatch* predicate.

$\langle 4\rangle 7.$ QED
BY $\langle 3\rangle 11$, $\langle 4\rangle 1$, $\langle 4\rangle 2$, $\langle 4\rangle 3$, $\langle 4\rangle 4$, $\langle 4\rangle 5$, $\langle 4\rangle 6$
DEF *AuthenticatorsMatch*, *ll1NewHistoryStateBinding*,
ll2NewHistorySummaryHash, *ll2NewHistoryStateBinding*

The remainder of the proof for *LL2PerformOperation* is a series of assertions, one for each conjunct in the definition of the *LL1PerformOperation* action.

The first conjunct in *LL1PerformOperation*. This is basically just an application of *AuthenticatorValidatedLemma*

$\langle 3\rangle 13.\ ValidateMAC(LL1NVRAM.symmetricKey, ll1HistoryStateBinding, LL1RAM.authenticator)$

We need the fact that the symmetric keys in the *NVRAM* are equal across the two specs.

$\langle 4\rangle 1.\ LL1NVRAM.symmetricKey = LL2NVRAM.symmetricKey$
BY $\langle 2\rangle 1$ DEF *LL2Refinement*

We prove the types that are needed for *AuthenticatorValidatedLemma*.

$\langle 4\rangle 2.\ ll2StateHash \in HashType$
BY $\langle 3\rangle 3$

$\langle 4\rangle 3.\ LL1RAM.historySummary \in HashType$
BY $\langle 2\rangle 1$ DEF *LL2Refinement*, *LL1UntrustedStorageType*

$\langle 4\rangle 4.\ LL2RAM.historySummary \in HistorySummaryType$
BY $\langle 2\rangle 1$, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*

⟨4⟩5. $LL1RAM.authenticator \in MACType$
BY ⟨2⟩1 DEF $LL2Refinement$, $LL1UntrustedStorageType$
⟨4⟩6. $LL2RAM.authenticator \in MACType$
BY ⟨2⟩1, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$
⟨4⟩7. $LL2NVRAM.symmetricKey \in SymmetricKeyType$
BY ⟨2⟩1, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$
⟨4⟩8. $LL2NVRAM.hashBarrier \in HashType$
BY ⟨2⟩1, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$

There are three preconditions for $AuthenticatorValidatedLemma$. The first precondition follows from the refinement.

⟨4⟩9. $HistorySummariesMatch($
$LL1RAM.historySummary, LL2RAM.historySummary, LL2NVRAM.hashBarrier)$
BY ⟨2⟩1 DEF $LL2Refinement$

The second precondition also follows from the refinement.

⟨4⟩10. PICK $symmetricKey \in SymmetricKeyType$:
$AuthenticatorsMatch($
$LL1RAM.authenticator,$
$LL2RAM.authenticator,$
$symmetricKey,$
$LL2NVRAM.hashBarrier)$
BY ⟨2⟩1 DEF $LL2Refinement$

The third precondition follows from the definition of the $LL2PerformOperation$ action.

⟨4⟩11. $ValidateMAC(LL2NVRAM.symmetricKey, ll2HistoryStateBinding, LL2RAM.authenticator)$
BY ⟨3⟩2 DEF $ll2HistoryStateBinding$, $ll2StateHash$, $ll2HistorySummaryHash$
⟨4⟩12. QED

Ideally, this QED step should just read:

BY ⟨3⟩6, ⟨4⟩1, ⟨4⟩2, ⟨4⟩3, ⟨4⟩4, ⟨4⟩5, ⟨4⟩6, ⟨4⟩7, ⟨4⟩8, ⟨4⟩9, ⟨4⟩10, ⟨4⟩11,
$AuthenticatorValidatedLemma$
DEF $ll1HistoryStateBinding$, $ll2HistoryStateBinding$, $ll2HistorySummaryHash$

However, the prover seems to get a little confused in this instance. We make life easier for the prover by explicitly staging the instantiation of the quantified variables within the definition of $AuthenticatorValidatedLemma$.

⟨5⟩1. $\forall$ $samLL2Authenticator \in MACType,$
$samSymmetricKey2 \in SymmetricKeyType,$
$samHashBarrier \in HashType$:
$AuthenticatorValidatedLemma!(ll2StateHash, LL1RAM.historySummary,$
$LL2RAM.historySummary, LL1RAM.authenticator, samLL2Authenticator,$
$symmetricKey, samSymmetricKey2, samHashBarrier)!1$
BY ⟨4⟩2, ⟨4⟩3, ⟨4⟩4, ⟨4⟩5, $AuthenticatorValidatedLemma$
⟨5⟩2. $AuthenticatorValidatedLemma!(ll2StateHash, LL1RAM.historySummary,$
$LL2RAM.historySummary, LL1RAM.authenticator, LL2RAM.authenticator,$
$symmetricKey, LL2NVRAM.symmetricKey, LL2NVRAM.hashBarrier)!1$
BY ⟨4⟩2, ⟨4⟩3, ⟨4⟩4, ⟨4⟩5, ⟨4⟩6, ⟨4⟩7, ⟨4⟩8, ⟨4⟩10, ⟨5⟩1
⟨5⟩3. QED
BY ⟨3⟩6, ⟨4⟩1, ⟨4⟩2, ⟨4⟩3, ⟨4⟩4, ⟨4⟩5, ⟨4⟩6, ⟨4⟩7, ⟨4⟩8, ⟨4⟩9, ⟨4⟩10, ⟨4⟩11, ⟨5⟩2
DEF $ll1HistoryStateBinding$, $ll2HistoryStateBinding$, $ll2HistorySummaryHash$

The second conjunct in $LL1PerformOperation$: In the Memoir-Basic spec, the history summary in the $NVRAM$ equals the history summary in the RAM.

⟨3⟩14. $LL1NVRAM.historySummary = LL1RAM.historySummary$

The history summaries in the $NVRAM$ match across the two specs.

⟨4⟩1. $HistorySummariesMatch($

$LL1NVRAM.historySummary$,
$LL2NVRAMLogicalHistorySummary$,
$LL2NVRAM.hashBarrier$)
BY $\langle 2\rangle 1$ DEF $LL2Refinement$

The history summaries in the RAM match across the two specs.

$\langle 4\rangle 2$. $HistorySummariesMatch($
$LL1RAM.historySummary$, $LL2RAM.historySummary$, $LL2NVRAM.hashBarrier$)
BY $\langle 2\rangle 1$ DEF $LL2Refinement$

In the Memoir-Opt spec, we separately handle the two cases of whether an extension is in progress. If an extension is in progress, then the logical history summary in the *NVRAM* equals the history summary in the RAM, so we can use the *HistorySummariesMatchUniqueLemma*.

$\langle 4\rangle 3$. CASE $LL2NVRAM.extensionInProgress$
$\langle 5\rangle 1$. $LL2NVRAMLogicalHistorySummary = LL2RAM.historySummary$
$\langle 6\rangle 1$. $LL2NVRAMLogicalHistorySummary = ll2CurrentHistorySummary$
$\langle 7\rangle 1$. $LL2NVRAMLogicalHistorySummary = [$
$anchor \mapsto LL2NVRAM.historySummaryAnchor$,
$extension \mapsto LL2SPCR]$
$\langle 8\rangle 1$. $LL2SPCR \neq BaseHashValue$
BY $\langle 3\rangle 2$, $\langle 4\rangle 3$
$\langle 8\rangle 2$. QED
BY $\langle 4\rangle 3$, $\langle 8\rangle 1$ DEF $LL2NVRAMLogicalHistorySummary$
$\langle 7\rangle 2$. $ll2CurrentHistorySummary = [$
$anchor \mapsto LL2NVRAM.historySummaryAnchor$,
$extension \mapsto LL2SPCR]$
BY DEF $ll2CurrentHistorySummary$
$\langle 7\rangle 3$. QED
BY $\langle 7\rangle 1$, $\langle 7\rangle 2$
$\langle 6\rangle 2$. $ll2CurrentHistorySummary = LL2RAM.historySummary$
BY $\langle 3\rangle 2$, $\langle 4\rangle 3$ DEF $ll2CurrentHistorySummary$
$\langle 6\rangle 3$. QED
BY $\langle 6\rangle 1$, $\langle 6\rangle 2$

We use the *HistorySummariesMatchUniqueLemma* to prove that the fields are equal. This requires proving some types.

$\langle 5\rangle 2$. QED
$\langle 6\rangle 1$. $LL1NVRAM.historySummary \in HashType$
BY $\langle 2\rangle 1$ DEF $LL2Refinement$, $LL1TrustedStorageType$
$\langle 6\rangle 2$. $LL1RAM.historySummary \in HashType$
BY $\langle 2\rangle 1$ DEF $LL2Refinement$, $LL1UntrustedStorageType$
$\langle 6\rangle 3$. $LL2RAM.historySummary \in HistorySummaryType$
BY $\langle 2\rangle 1$, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$
$\langle 6\rangle 4$. $LL2NVRAM.hashBarrier \in HashType$
BY $\langle 2\rangle 1$, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$
$\langle 6\rangle 5$. QED
BY $\langle 4\rangle 1$, $\langle 4\rangle 2$, $\langle 5\rangle 1$, $\langle 6\rangle 1$, $\langle 6\rangle 2$, $\langle 6\rangle 3$, $\langle 6\rangle 4$, $HistorySummariesMatchUniqueLemma$

If an extension is not in progress, then the logical history summary in the *NVRAM* is a checkpoint of the history summary in the RAM, so we can use the *HistorySummariesMatchAcrossCheckpointLemma*.

$\langle 4\rangle 4$. CASE $\neg LL2NVRAM.extensionInProgress$
$\langle 5\rangle 1$. $LL2NVRAMLogicalHistorySummary = Checkpoint(LL2RAM.historySummary)$
$\langle 6\rangle 1$. $LL2NVRAMLogicalHistorySummary = ll2CurrentHistorySummary$
$\langle 7\rangle 1$. $LL2NVRAMLogicalHistorySummary = [$
$anchor \mapsto LL2NVRAM.historySummaryAnchor$,
$extension \mapsto BaseHashValue]$

BY ⟨4⟩4 DEF *LL2NVRAMLogicalHistorySummary*

⟨7⟩2. $ll2CurrentHistorySummary = [$
$anchor \mapsto LL2NVRAM.historySummaryAnchor,$
$extension \mapsto LL2SPCR]$

BY DEF *ll2CurrentHistorySummary*

⟨7⟩3. $LL2SPCR = BaseHashValue$

BY ⟨3⟩2, ⟨4⟩4

⟨7⟩4. QED

BY ⟨7⟩1, ⟨7⟩2, ⟨7⟩3

⟨6⟩2. $ll2CurrentHistorySummary = Checkpoint(LL2RAM.historySummary)$

BY ⟨3⟩2, ⟨4⟩4 DEF *ll2CurrentHistorySummary*

⟨6⟩3. QED

BY ⟨6⟩1, ⟨6⟩2

We use the *HistorySummariesMatchAcrossCheckpointLemma* to prove that the fields are equal. This requires proving some types.

⟨5⟩2. QED

⟨6⟩1. $LL1NVRAM.historySummary \in HashType$

BY ⟨2⟩1 DEF *LL2Refinement*, *LL1TrustedStorageType*

⟨6⟩2. $LL1RAM.historySummary \in HashType$

BY ⟨2⟩1 DEF *LL2Refinement*, *LL1UntrustedStorageType*

⟨6⟩3. $LL2NVRAMLogicalHistorySummary \in HistorySummaryType$

BY ⟨2⟩1, *LL2NVRAMLogicalHistorySummaryTypeSafe*

⟨6⟩4. $LL2RAM.historySummary \in HistorySummaryType$

BY ⟨2⟩1, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*

⟨6⟩5. $LL2NVRAM.hashBarrier \in HashType$

BY ⟨2⟩1, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*

⟨6⟩6. QED

BY ⟨4⟩1, ⟨4⟩2, ⟨5⟩1, ⟨6⟩1, ⟨6⟩2, ⟨6⟩3, ⟨6⟩4, ⟨6⟩5, *HistorySummariesMatchAcrossCheckpointLemma*

⟨4⟩5. QED

BY ⟨4⟩3, ⟨4⟩4

The third conjunct in *LL1PerformOperation*. This conjunct asserts a record equality, so we prove each field in the record separately.

⟨3⟩15. $LL1RAM' = [$
$publicState \mapsto ll1sResult.newPublicState,$
$privateStateEnc \mapsto ll1NewPrivateStateEnc,$
$historySummary \mapsto ll1NewHistorySummary,$
$authenticator \mapsto ll1NewAuthenticator]$

The public state field in the primed Memoir-Basic RAM equals the public state in the result of the service, because the primed RAM variables match across the two specs.

⟨4⟩1. $LL1RAM.publicState' = ll1sResult.newPublicState$

⟨5⟩1. $LL1RAM.publicState' = LL2RAM.publicState'$

BY ⟨2⟩1 DEF *LL2Refinement*

⟨5⟩2. $LL2RAM.publicState' = ll2SResult.newPublicState$

BY ⟨3⟩2 DEF *ll2SResult*, *ll2PrivateState*

⟨5⟩3. $ll1sResult.newPublicState = ll2SResult.newPublicState$

BY ⟨3⟩8

⟨5⟩4. QED

BY ⟨5⟩1, ⟨5⟩2, ⟨5⟩3

The encrypted private state field in the primed Memoir-Basic RAM equals the encrypted private state in the result of the service, because the primed RAM variables match across the two specs.

⟨4⟩2. $LL1RAM.privateStateEnc' = ll1NewPrivateStateEnc$

⟨5⟩1. $LL1RAM.privateStateEnc' = LL2RAM.privateStateEnc'$

BY $\langle 2 \rangle 1$ DEF *LL2Refinement*

$\langle 5 \rangle 2$. $LL2RAM.privateStateEnc' = ll2NewPrivateStateEnc$

BY $\langle 3 \rangle 2$ DEF *ll2NewPrivateStateEnc*, *ll2SResult*, *ll2PrivateState*

$\langle 5 \rangle 3$. $ll1NewPrivateStateEnc = ll2NewPrivateStateEnc$

BY $\langle 3 \rangle 9$

$\langle 5 \rangle 4$. QED

BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$, $\langle 5 \rangle 3$

The history summary field in the primed Memoir-Basic RAM equals the new history summary defined in the *LL1PerformOperation* action, because the history summaries in the RAM variables match across the specs by refinement. We use the *HistorySummariesMatchUniqueLemma* to prove the equality.

$\langle 4 \rangle 3$. $LL1RAM.historySummary' = ll1NewHistorySummary$

$\langle 5 \rangle 1$. $HistorySummariesMatch($
$LL1RAM.historySummary', LL2RAM.historySummary', LL2NVRAM.hashBarrier')$

BY $\langle 2 \rangle 1$ DEF *LL2Refinement*

$\langle 5 \rangle 2$. $LL2RAM.historySummary' = ll2NewHistorySummary$

BY $\langle 3 \rangle 2$ DEF *ll2NewHistorySummary*, *ll2CurrentHistorySummary*

$\langle 5 \rangle 3$. $HistorySummariesMatch($
$ll1NewHistorySummary, ll2NewHistorySummary, LL2NVRAM.hashBarrier')$

BY $\langle 3 \rangle 10$

We use the *HistorySummariesMatchUniqueLemma* to prove that the fields are equal. This requires proving some types.

$\langle 5 \rangle 4$. QED

$\langle 6 \rangle 1$. $LL1RAM.historySummary' \in HashType$

BY $\langle 2 \rangle 1$ DEF *LL2Refinement*, *LL1UntrustedStorageType*

$\langle 6 \rangle 2$. $ll1NewHistorySummary \in HashType$

BY $\langle 3 \rangle 5$

$\langle 6 \rangle 3$. $LL2RAM.historySummary' \in HistorySummaryType$

BY $\langle 2 \rangle 1$, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*

$\langle 6 \rangle 4$. $LL2NVRAM.hashBarrier' \in HashType$

BY $\langle 2 \rangle 1$, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*

$\langle 6 \rangle 5$. QED

BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$, $\langle 5 \rangle 3$, $\langle 6 \rangle 1$, $\langle 6 \rangle 2$, $\langle 6 \rangle 3$, $\langle 6 \rangle 4$, *HistorySummariesMatchUniqueLemma*

The authenticator field in the primed Memoir-Basic RAM equals the new authenticator defined in the *LL1PerformOperation* action, because the authenticators in the RAM variables match across the specs by refinement. We use the *AuthenticatorsMatchUniqueLemma* to prove the equality.

$\langle 4 \rangle 4$. $LL1RAM.authenticator' = ll1NewAuthenticator$

$\langle 5 \rangle 1$. PICK $symmetricKey \in SymmetricKeyType$:
$AuthenticatorsMatch($
$LL1RAM.authenticator',$
$LL2RAM.authenticator',$
$symmetricKey,$
$LL2NVRAM.hashBarrier')$

BY $\langle 2 \rangle 1$ DEF *LL2Refinement*

$\langle 5 \rangle 2$. $LL2RAM.authenticator' = ll2NewAuthenticator$

BY $\langle 3 \rangle 2$

DEF *ll2NewAuthenticator*, *ll2NewHistoryStateBinding*, *ll2NewHistorySummaryHash*, *ll2NewStateHash*, *ll2NewPrivateStateEnc*, *ll2SResult*, *ll2PrivateState*, *ll2NewHistorySummary*, *ll2CurrentHistorySummary*

$\langle 5 \rangle 3$. $AuthenticatorsMatch($
$ll1NewAuthenticator,$
$ll2NewAuthenticator,$
$LL2NVRAM.symmetricKey',$

$LL2NVRAM.hashBarrier'$)

BY ⟨3⟩12

We use the *AuthenticatorsMatchUniqueLemma* to prove that the fields are equal. This requires proving some types.

⟨5⟩4. QED

⟨6⟩1. $LL1RAM.authenticator' \in MACType$

BY ⟨2⟩1 DEF *LL2Refinement*, *LL1UntrustedStorageType*

⟨6⟩2. $ll1NewAuthenticator \in MACType$

BY ⟨3⟩5

⟨6⟩3. $LL2RAM.authenticator' \in MACType$

BY ⟨2⟩1, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*

⟨6⟩4. $symmetricKey \in SymmetricKeyType$

BY ⟨5⟩1

⟨6⟩5. $LL2NVRAM.symmetricKey' \in SymmetricKeyType$

BY ⟨2⟩1, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*

⟨6⟩6. $LL2NVRAM.hashBarrier' \in HashType$

BY ⟨2⟩1, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*

⟨6⟩7. QED

BY ⟨5⟩1, ⟨5⟩2, ⟨5⟩3, ⟨6⟩1, ⟨6⟩2, ⟨6⟩3, ⟨6⟩4, ⟨6⟩5, ⟨6⟩6, *AuthenticatorsMatchUniqueLemma*

The refinement asserts that the RAM record has the appropriate type.

⟨4⟩5. $LL1RAM' \in LL1UntrustedStorageType$

BY ⟨2⟩1 DEF *LL2Refinement*

We use the *LL1RAMRecordCompositionLemma* to unify the field equalities into a record equality.

⟨4⟩6. QED

BY ⟨4⟩1, ⟨4⟩2, ⟨4⟩3, ⟨4⟩4, ⟨4⟩5, *LL1RAMRecordCompositionLemma*

The fourth conjunct in *LL1PerformOperation*. This conjunct asserts a record equality, so we prove each field in the record separately.

⟨3⟩16. $LL1NVRAM' = [$
$historySummary \mapsto ll1NewHistorySummary,$
$symmetricKey \mapsto LL1NVRAM.symmetricKey]$

The history summary field in the primed Memoir-Basic *NVRAM* equals the new history summary defined in the *LL1PerformOperation* action, because the logical history summary in the Memoir-Opt *NVRAM* and *SPCR* matches the history summary in the Memoir-Basic *NVRAM* by refinement. We use the *HistorySummariesMatchUniqueLemma* to prove the equality.

⟨4⟩1. $LL1NVRAM.historySummary' = ll1NewHistorySummary$

⟨5⟩1. $HistorySummariesMatch($
$LL1NVRAM.historySummary',$
$LL2NVRAMLogicalHistorySummary',$
$LL2NVRAM.hashBarrier')$

BY ⟨2⟩1 DEF *LL2Refinement*

In the Memoir-Opt spec, the logical history summary equals the new history summary defined in the *LL2PerformOperation* action. Proving this is slightly involved, because we have to show that the updates performed by *LL2PerformOperation* are logically equivalent to the operation of the *Successor* operator.

⟨5⟩2. $LL2NVRAMLogicalHistorySummary' = ll2NewHistorySummary$

⟨6⟩1. $LL2NVRAMLogicalHistorySummary' = [$
$anchor \mapsto LL2NVRAM.historySummaryAnchor',$
$extension \mapsto LL2SPCR']$

⟨7⟩1. $LL2NVRAM.extensionInProgress' = \text{TRUE}$

BY ⟨3⟩2

⟨7⟩2. $LL2SPCR' \neq BaseHashValue$

$\langle 8\rangle 1.\ LL2SPCR' = ll2NewHistorySummary.extension$
BY $\langle 3\rangle 2$ DEF $ll2NewHistorySummary$, $ll2CurrentHistorySummary$
$\langle 8\rangle 2.\ ll2NewHistorySummary.extension \neq BaseHashValue$
$\langle 9\rangle 1.\ ll2CurrentHistorySummary \in HistorySummaryType$
BY $\langle 3\rangle 3$
$\langle 9\rangle 2.\ input \in InputType$
$\langle 10\rangle 1.\ input \in LL2AvailableInputs$
BY $\langle 3\rangle 2$
$\langle 10\rangle 2.\ LL2AvailableInputs \subseteq InputType$
BY $\langle 2\rangle 1$ DEF $LL2TypeInvariant$
$\langle 10\rangle 3.$ QED
BY $\langle 10\rangle 1$, $\langle 10\rangle 2$
$\langle 9\rangle 3.\ LL2NVRAM.hashBarrier \in HashType$
BY $\langle 2\rangle 1$, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$
$\langle 9\rangle 4.$ QED
BY $\langle 9\rangle 1$, $\langle 9\rangle 2$, $\langle 9\rangle 3$, $SuccessorHasNonBaseExtensionLemma$
DEF $ll2NewHistorySummary$
$\langle 8\rangle 3.$ QED
BY $\langle 8\rangle 1$, $\langle 8\rangle 2$
$\langle 7\rangle 3.$ QED
BY $\langle 7\rangle 1$, $\langle 7\rangle 2$ DEF $LL2NVRAMLogicalHistorySummary$
$\langle 6\rangle 2.\ ll2NewHistorySummary = [$
$anchor \mapsto LL2NVRAM.historySummaryAnchor'$,
$extension \mapsto LL2SPCR']$
$\langle 7\rangle 1.\ ll2NewHistorySummary.anchor = LL2NVRAM.historySummaryAnchor'$
$\langle 8\rangle 1.\ ll2NewHistorySummary =$
$Successor(ll2CurrentHistorySummary, input, LL2NVRAM.hashBarrier)$
BY DEF $ll2NewHistorySummary$
$\langle 8\rangle 2.\ ll2NewHistorySummary.anchor = ll2CurrentHistorySummary.anchor$
BY $\langle 8\rangle 1$ DEF $Successor$
$\langle 8\rangle 3.\ ll2NewHistorySummary.anchor = LL2NVRAM.historySummaryAnchor$
BY $\langle 8\rangle 2$ DEF $ll2CurrentHistorySummary$
$\langle 8\rangle 4.$ UNCHANGED $LL2NVRAM.historySummaryAnchor$
BY $\langle 3\rangle 2$
$\langle 8\rangle 5.$ QED
BY $\langle 8\rangle 3$, $\langle 8\rangle 4$
$\langle 7\rangle 2.\ ll2NewHistorySummary.extension = LL2SPCR'$
BY $\langle 3\rangle 2$ DEF $ll2NewHistorySummary$, $ll2CurrentHistorySummary$
$\langle 7\rangle 3.\ ll2NewHistorySummary \in HistorySummaryType$
BY $\langle 3\rangle 3$
$\langle 7\rangle 4.$ QED
BY $\langle 7\rangle 1$, $\langle 7\rangle 2$, $\langle 7\rangle 3$, $HistorySummaryRecordCompositionLemma$
$\langle 6\rangle 3.$ QED
BY $\langle 6\rangle 1$, $\langle 6\rangle 2$
$\langle 5\rangle 3.\ HistorySummariesMatch($
$ll1NewHistorySummary, ll2NewHistorySummary, LL2NVRAM.hashBarrier')$
BY $\langle 3\rangle 10$

We use the *HistorySummariesMatchUniqueLemma* to prove that the fields are equal. This requires proving some types.

$\langle 5\rangle 4.$ QED
$\langle 6\rangle 1.\ LL1NVRAM.historySummary' \in HashType$
BY $\langle 2\rangle 1$ DEF $LL2Refinement$, $LL1TrustedStorageType$

$\langle 6\rangle 2.$ $ll1NewHistorySummary \in HashType$
BY $\langle 3\rangle 5$
$\langle 6\rangle 3.$ $LL2NVRAMLogicalHistorySummary' \in HistorySummaryType$
BY $\langle 2\rangle 1$, $LL2NVRAMLogicalHistorySummaryTypeSafe$
$\langle 6\rangle 4.$ $LL2NVRAM.hashBarrier' \in HashType$
BY $\langle 2\rangle 1$, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$
$\langle 6\rangle 5.$ QED
BY $\langle 5\rangle 1$, $\langle 5\rangle 2$, $\langle 5\rangle 3$, $\langle 6\rangle 1$, $\langle 6\rangle 2$, $\langle 6\rangle 3$, $\langle 6\rangle 4$, $HistorySummariesMatchUniqueLemma$

The symmetric key in the $NVRAM$ is unchanged by the $UnchangedNVRAMSymmetricKeyLemma$.

$\langle 4\rangle 2.$ UNCHANGED $LL1NVRAM.symmetricKey$
BY $\langle 2\rangle 1$, $\langle 3\rangle 2$, $UnchangedNVRAMSymmetricKeyLemma$

The refinement asserts that the $NVRAM$ record has the appropriate type.

$\langle 4\rangle 3.$ $LL1NVRAM' \in LL1TrustedStorageType$
BY $\langle 2\rangle 1$ DEF $LL2Refinement$

We use the $LL1NVRAMRecordCompositionLemma$ to unify the field equalities into a record equality.

$\langle 4\rangle 4.$ QED
BY $\langle 4\rangle 1$, $\langle 4\rangle 2$, $\langle 4\rangle 3$, $LL1NVRAMRecordCompositionLemma$

The fifth conjunct in $LL1PerformOperation$. The set of observed outputs is equal across the two specs.

$\langle 3\rangle 17.$ $LL1ObservedOutputs' = LL1ObservedOutputs \cup \{ll1sResult.output\}$
$\langle 4\rangle 1.$ $LL1ObservedOutputs = LL2ObservedOutputs$
BY $\langle 2\rangle 1$ DEF $LL2Refinement$
$\langle 4\rangle 2.$ $LL1ObservedOutputs' = LL2ObservedOutputs'$
BY $\langle 2\rangle 1$ DEF $LL2Refinement$
$\langle 4\rangle 3.$ $ll1sResult.output = ll2SResult.output$
BY $\langle 3\rangle 8$
$\langle 4\rangle 4.$ $LL2ObservedOutputs' = LL2ObservedOutputs \cup \{ll2SResult.output\}$
BY $\langle 3\rangle 2$ DEF $ll2SResult$, $ll2PrivateState$
$\langle 4\rangle 5.$ QED
BY $\langle 4\rangle 1$, $\langle 4\rangle 2$, $\langle 4\rangle 3$, $\langle 4\rangle 4$

The sixth conjunct in $LL1PerformOperation$. The disk is unchanged by the $UnchangedDiskLemma$.

$\langle 3\rangle 18.$ UNCHANGED $LL1Disk$
BY $\langle 2\rangle 1$, $\langle 3\rangle 2$, $UnchangedDiskLemma$

The seventh conjunct in $LL1PerformOperation$. The set of available inputs is unchanged by the $UnchangedAvailableInputsLemma$.

$\langle 3\rangle 19.$ UNCHANGED $LL1AvailableInputs$
BY $\langle 2\rangle 1$, $\langle 3\rangle 2$, $UnchangedAvailableInputsLemma$

The eighth conjunct in $LL1PerformOperation$. We prove that the primed set of observed authenticators matches across the specs, that the unprimed set of observed authenticators matches across the specs, and that the new authenticator matches across the specs. The $AuthenticatorSetsMatchUniqueLemma$ then proves the equality.

$\langle 3\rangle 20.$ $LL1ObservedAuthenticators' =$
$LL1ObservedAuthenticators \cup \{ll1NewAuthenticator\}$

The primed set of observed authenticators matches across the specs. This follows directly from the refinement.

$\langle 4\rangle 1.$ $AuthenticatorSetsMatch($
$LL1ObservedAuthenticators'$,
$LL2ObservedAuthenticators'$,
$LL2NVRAM.symmetricKey'$,
$LL2NVRAM.hashBarrier')$
BY $\langle 2\rangle 1$ DEF $LL2Refinement$

The union matches across the specs. We prove this by proving the matching of each constituent set.

$\langle 4\rangle 2.$ $AuthenticatorSetsMatch($

$LL1ObservedAuthenticators \cup \{ll1NewAuthenticator\}$,
$LL2ObservedAuthenticators \cup \{ll2NewAuthenticator\}$,
$LL2NVRAM.symmetricKey'$,
$LL2NVRAM.hashBarrier'$)

⟨5⟩1. UNCHANGED ⟨$LL2NVRAM.symmetricKey$, $LL2NVRAM.hashBarrier$⟩
BY ⟨3⟩2

The unprimed set of observed authenticators matches across the specs. This follows directly from the refinement.

⟨5⟩2. $AuthenticatorSetsMatch($
$LL1ObservedAuthenticators$,
$LL2ObservedAuthenticators$,
$LL2NVRAM.symmetricKey'$,
$LL2NVRAM.hashBarrier'$)
BY ⟨2⟩1, ⟨5⟩1 DEF $LL2Refinement$

The new authenticator matches across the specs. This follows directly from the refinement.

⟨5⟩3. $AuthenticatorsMatch($
$ll1NewAuthenticator$,
$ll2NewAuthenticator$,
$LL2NVRAM.symmetricKey'$,
$LL2NVRAM.hashBarrier'$)
BY ⟨2⟩1, ⟨5⟩1 DEF $LL2Refinement$

⟨5⟩4. QED
BY ⟨5⟩2, ⟨5⟩3 DEF $AuthenticatorSetsMatch$

In the Memoir-Opt spec, the primed set of observed authenticators is formed from the union of the unprimed set of observed authenticators and the new authenticator.

⟨4⟩3. $LL2ObservedAuthenticators' =$
$LL2ObservedAuthenticators \cup \{ll2NewAuthenticator\}$
BY ⟨3⟩2
DEF $ll2NewAuthenticator$, $ll2NewHistoryStateBinding$, $ll2NewHistorySummaryHash$, $ll2NewStateHash$, $ll2NewPrivateStateEnc$, $ll2SResult$, $ll2PrivateState$, $ll2NewHistorySummary$, $ll2CurrentHistorySummary$

⟨4⟩4. QED

We use the $AuthenticatorSetsMatchUniqueLemma$ to prove that the sets are equal. This requires proving some types.

⟨5⟩1. $LL1ObservedAuthenticators' \in$ SUBSET $MACType$
BY ⟨2⟩1 DEF $LL2Refinement$, $LL1UntrustedStorageType$

⟨5⟩2. $LL1ObservedAuthenticators \cup \{ll1NewAuthenticator\} \in$
SUBSET $MACType$

⟨6⟩1. $LL1ObservedAuthenticators \in$ SUBSET $MACType$
BY ⟨2⟩1 DEF $LL2Refinement$, $LL1UntrustedStorageType$

⟨6⟩2. $ll1NewAuthenticator \in MACType$
BY ⟨3⟩5

⟨6⟩3. QED
BY ⟨6⟩1, ⟨6⟩2

⟨5⟩3. $LL2ObservedAuthenticators' \in$ SUBSET $MACType$
BY ⟨2⟩1 DEF $LL2TypeInvariant$

⟨5⟩4. $LL2NVRAM.symmetricKey' \in SymmetricKeyType$
BY ⟨2⟩1, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$

⟨5⟩5. $LL2NVRAM.hashBarrier' \in HashType$
BY ⟨2⟩1, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$

⟨5⟩6. QED
BY ⟨4⟩1, ⟨4⟩2, ⟨4⟩3, ⟨5⟩1, ⟨5⟩2, ⟨5⟩3, ⟨5⟩4, ⟨5⟩5,

$AuthenticatorSetsMatchUniqueLemma$

$\langle 3 \rangle 21.$ QED

BY $\langle 3 \rangle 4$, $\langle 3 \rangle 13$, $\langle 3 \rangle 14$, $\langle 3 \rangle 15$, $\langle 3 \rangle 16$, $\langle 3 \rangle 17$, $\langle 3 \rangle 18$, $\langle 3 \rangle 19$, $\langle 3 \rangle 20$

DEF $LL1PerformOperation$, $ll1StateHash$, $ll1HistoryStateBinding$, $ll1PrivateState$, $ll1sResult$, $ll1NewPrivateStateEnc$, $ll1NewHistorySummary$, $ll1NewStateHash$, $ll1NewHistoryStateBinding$, $ll1NewAuthenticator$

A Memoir-Opt *LL2RepeatOperation* action refines to a Memoir-Basic *LL1RepeatOperation* action.

$\langle 2 \rangle 5.$ $LL2RepeatOperation \Rightarrow LL1RepeatOperation$

We assume the antecedent.

$\langle 3 \rangle 1.$ HAVE $LL2RepeatOperation$

We pick an input that satisfies the *LL2RepeatOperation* predicate.

$\langle 3 \rangle 2.$ PICK $input \in LL2AvailableInputs : LL2RepeatOperation!(input)!1$

BY $\langle 3 \rangle 1$ DEF $LL2RepeatOperation$

We re-state the definitions from *LL2RepeatOperation*.

$\langle 3 \rangle$ $ll2HistorySummaryHash \triangleq$
$Hash(LL2RAM.historySummary.anchor, LL2RAM.historySummary.extension)$

$\langle 3 \rangle$ $ll2StateHash \triangleq Hash(LL2RAM.publicState, LL2RAM.privateStateEnc)$

$\langle 3 \rangle$ $ll2HistoryStateBinding \triangleq Hash(ll2HistorySummaryHash, ll2StateHash)$

$\langle 3 \rangle$ $ll2NewHistorySummary \triangleq Successor(LL2RAM.historySummary, input, LL2NVRAM.hashBarrier)$

$\langle 3 \rangle$ $ll2CheckpointedHistorySummary \triangleq Checkpoint(LL2RAM.historySummary)$

$\langle 3 \rangle$ $ll2NewCheckpointedHistorySummary \triangleq$
$Successor(ll2CheckpointedHistorySummary, input, LL2NVRAM.hashBarrier)$

$\langle 3 \rangle$ $ll2CheckpointedNewHistorySummary \triangleq Checkpoint(ll2NewHistorySummary)$

$\langle 3 \rangle$ $ll2CheckpointedNewCheckpointedHistorySummary \triangleq$
$Checkpoint(ll2NewCheckpointedHistorySummary)$

$\langle 3 \rangle$ $ll2PrivateState \triangleq SymmetricDecrypt(LL2NVRAM.symmetricKey, LL2RAM.privateStateEnc)$

$\langle 3 \rangle$ $ll2SResult \triangleq Service(LL2RAM.publicState, ll2PrivateState, input)$

$\langle 3 \rangle$ $ll2NewPrivateStateEnc \triangleq$
$SymmetricEncrypt(LL2NVRAM.symmetricKey, ll2SResult.newPrivateState)$

$\langle 3 \rangle$ $ll2CurrentHistorySummary \triangleq [$
$anchor \mapsto LL2NVRAM.historySummaryAnchor,$
$extension \mapsto LL2SPCR]$

$\langle 3 \rangle$ $ll2CurrentHistorySummaryHash \triangleq Hash(LL2NVRAM.historySummaryAnchor, LL2SPCR)$

$\langle 3 \rangle$ $ll2NewStateHash \triangleq Hash(ll2SResult.newPublicState, ll2NewPrivateStateEnc)$

$\langle 3 \rangle$ $ll2NewHistoryStateBinding \triangleq Hash(ll2CurrentHistorySummaryHash, ll2NewStateHash)$

$\langle 3 \rangle$ $ll2NewAuthenticator \triangleq GenerateMAC(LL2NVRAM.symmetricKey, ll2NewHistoryStateBinding)$

We prove that the definitions from *LL2RepeatOperation* satisfy their types, using the *LL2RepeatOperationDefsTypeSafeLemma*.

$\langle 3 \rangle 3.$ $\land$ $ll2HistorySummaryHash \in HashType$
$\land$ $ll2StateHash \in HashType$
$\land$ $ll2HistoryStateBinding \in HashType$
$\land$ $ll2NewHistorySummary \in HistorySummaryType$
$\land$ $ll2NewHistorySummary.anchor \in HashType$
$\land$ $ll2NewHistorySummary.extension \in HashType$
$\land$ $ll2CheckpointedHistorySummary \in HistorySummaryType$
$\land$ $ll2CheckpointedHistorySummary.anchor \in HashType$
$\land$ $ll2CheckpointedHistorySummary.extension \in HashType$
$\land$ $ll2NewCheckpointedHistorySummary \in HistorySummaryType$
$\land$ $ll2NewCheckpointedHistorySummary.anchor \in HashType$
$\land$ $ll2NewCheckpointedHistorySummary.extension \in HashType$
$\land$ $ll2CheckpointedNewHistorySummary \in HistorySummaryType$

$\land$ *ll2CheckpointedNewHistorySummary.anchor* $\in$ *HashType*
$\land$ *ll2CheckpointedNewHistorySummary.extension* $\in$ *HashType*
$\land$ *ll2CheckpointedNewCheckpointedHistorySummary* $\in$ *HistorySummaryType*
$\land$ *ll2CheckpointedNewCheckpointedHistorySummary.anchor* $\in$ *HashType*
$\land$ *ll2CheckpointedNewCheckpointedHistorySummary.extension* $\in$ *HashType*
$\land$ *ll2PrivateState* $\in$ *PrivateStateType*
$\land$ *ll2SResult* $\in$ *ServiceResultType*
$\land$ *ll2SResult.newPublicState* $\in$ *PublicStateType*
$\land$ *ll2SResult.newPrivateState* $\in$ *PrivateStateType*
$\land$ *ll2SResult.output* $\in$ *OutputType*
$\land$ *ll2NewPrivateStateEnc* $\in$ *PrivateStateEncType*
$\land$ *ll2CurrentHistorySummary* $\in$ *HistorySummaryType*
$\land$ *ll2CurrentHistorySummary.anchor* $\in$ *HashType*
$\land$ *ll2CurrentHistorySummary.extension* $\in$ *HashType*
$\land$ *ll2CurrentHistorySummaryHash* $\in$ *HashType*
$\land$ *ll2NewStateHash* $\in$ *HashType*
$\land$ *ll2NewHistoryStateBinding* $\in$ *HashType*
$\land$ *ll2NewAuthenticator* $\in$ *MACType*

$\langle 4\rangle 1.$ *input* $\in$ *LL2AvailableInputs*
BY $\langle 3\rangle 2$
$\langle 4\rangle 2.$ *LL2TypeInvariant*
BY $\langle 2\rangle 1$
$\langle 4\rangle 3.$ QED
BY $\langle 4\rangle 1$, $\langle 4\rangle 2$, *LL2RepeatOperationDefsTypeSafeLemma*

We hide the definitions.

$\langle 3\rangle$ HIDE DEF *ll2HistorySummaryHash*, *ll2StateHash*, *ll2HistoryStateBinding*, *ll2NewHistorySummary*, *ll2CheckpointedHistorySummary*, *ll2NewCheckpointedHistorySummary*, *ll2CheckpointedNewHistorySummary*, *ll2CheckpointedNewCheckpointedHistorySummary*, *ll2PrivateState*, *ll2SResult*, *ll2NewPrivateStateEnc*, *ll2CurrentHistorySummary*, *ll2CurrentHistorySummaryHash*, *ll2NewStateHash*, *ll2NewHistoryStateBinding*, *ll2NewAuthenticator*

One fact that will be needed many places is that the input is in the Memoir-Basic set of available inputs.

$\langle 3\rangle 4.$ *input* $\in$ *LL1AvailableInputs*
$\langle 4\rangle 1.$ *input* $\in$ *LL2AvailableInputs*
BY $\langle 3\rangle 2$
$\langle 4\rangle 2.$ *LL1AvailableInputs* $=$ *LL2AvailableInputs*
BY $\langle 2\rangle 1$ DEF *LL2Refinement*
$\langle 4\rangle 3.$ QED
BY $\langle 4\rangle 1$, $\langle 4\rangle 2$

We re-state the definitions from *LL1RepeatOperation*.

$\langle 3\rangle$ *ll1StateHash* $\triangleq$ *Hash*(*LL1RAM.publicState*, *LL1RAM.privateStateEnc*)
$\langle 3\rangle$ *ll1HistoryStateBinding* $\triangleq$ *Hash*(*LL1RAM.historySummary*, *ll1StateHash*)
$\langle 3\rangle$ *ll1PrivateState* $\triangleq$ *SymmetricDecrypt*(*LL1NVRAM.symmetricKey*, *LL1RAM.privateStateEnc*)
$\langle 3\rangle$ *ll1sResult* $\triangleq$ *Service*(*LL1RAM.publicState*, *ll1PrivateState*, *input*)
$\langle 3\rangle$ *ll1NewPrivateStateEnc* $\triangleq$
SymmetricEncrypt(*LL1NVRAM.symmetricKey*, *ll1sResult.newPrivateState*)
$\langle 3\rangle$ *ll1NewStateHash* $\triangleq$ *Hash*(*ll1sResult.newPublicState*, *ll1NewPrivateStateEnc*)
$\langle 3\rangle$ *ll1NewHistoryStateBinding* $\triangleq$ *Hash*(*LL1NVRAM.historySummary*, *ll1NewStateHash*)
$\langle 3\rangle$ *ll1NewAuthenticator* $\triangleq$ *GenerateMAC*(*LL1NVRAM.symmetricKey*, *ll1NewHistoryStateBinding*)

We prove that the definitions from *LL1RepeatOperation* satisfy their types, using the *LL1RepeatOperationDefsTypeSafeLemma* and the *TypeSafetyRefinementLemma*.

$\langle 3 \rangle 5.$ $\wedge$ $ll1StateHash \in HashType$
$\wedge$ $ll1HistoryStateBinding \in HashType$
$\wedge$ $ll1PrivateState \in PrivateStateType$
$\wedge$ $ll1sResult \in ServiceResultType$
$\wedge$ $ll1sResult.newPublicState \in PublicStateType$
$\wedge$ $ll1sResult.newPrivateState \in PrivateStateType$
$\wedge$ $ll1sResult.output \in OutputType$
$\wedge$ $ll1NewPrivateStateEnc \in PrivateStateEncType$
$\wedge$ $ll1NewStateHash \in HashType$
$\wedge$ $ll1NewHistoryStateBinding \in HashType$
$\wedge$ $ll1NewAuthenticator \in MACType$

$\langle 4 \rangle 1.$ $input \in LL1AvailableInputs$
BY $\langle 3 \rangle 4$

$\langle 4 \rangle 2.$ $LL1TypeInvariant$
BY $\langle 2 \rangle 1$, $TypeSafetyRefinementLemma$

$\langle 4 \rangle 3.$ QED
BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$, $LL1RepeatOperationDefsTypeSafeLemma$

We hide the definitions from $LL1RepeatOperation$.

$\langle 3 \rangle$ HIDE DEF $ll1StateHash$, $ll1HistoryStateBinding$, $ll1PrivateState$, $ll1sResult$, $ll1NewPrivateStateEnc$, $ll1NewStateHash$, $ll1NewHistoryStateBinding$, $ll1NewAuthenticator$

We prove the correspondences between the definitions in $LL1RepeatOperation$ and $LL2RepeatOperation$. The state hashes are directly equal.

$\langle 3 \rangle 6.$ $ll1StateHash = ll2StateHash$

$\langle 4 \rangle 1.$ $LL1RAM.publicState = LL2RAM.publicState$
BY $\langle 2 \rangle 1$ DEF $LL2Refinement$

$\langle 4 \rangle 2.$ $LL1RAM.privateStateEnc = LL2RAM.privateStateEnc$
BY $\langle 2 \rangle 1$ DEF $LL2Refinement$

$\langle 4 \rangle 3.$ QED
BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$ DEF $ll1StateHash$, $ll2StateHash$

The private states are directly equal across the two specs.

$\langle 3 \rangle 7.$ $ll1PrivateState = ll2PrivateState$

$\langle 4 \rangle 1.$ $LL1NVRAM.symmetricKey = LL2NVRAM.symmetricKey$
BY $\langle 2 \rangle 1$ DEF $LL2Refinement$

$\langle 4 \rangle 2.$ $LL1RAM.privateStateEnc = LL2RAM.privateStateEnc$
BY $\langle 2 \rangle 1$ DEF $LL2Refinement$

$\langle 4 \rangle 3.$ QED
BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$ DEF $ll1PrivateState$, $ll2PrivateState$

The service results are directly equal across the two specs.

$\langle 3 \rangle 8.$ $ll1sResult = ll2SResult$

$\langle 4 \rangle 1.$ $LL1RAM.publicState = LL2RAM.publicState$
BY $\langle 2 \rangle 1$ DEF $LL2Refinement$

$\langle 4 \rangle 2.$ $ll1PrivateState = ll2PrivateState$
BY $\langle 3 \rangle 7$

$\langle 4 \rangle 3.$ QED
BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$ DEF $ll1sResult$, $ll2SResult$

The new encrypted private states are directly equal across the two specs.

$\langle 3 \rangle 9.$ $ll1NewPrivateStateEnc = ll2NewPrivateStateEnc$

$\langle 4 \rangle 1.$ $LL1NVRAM.symmetricKey = LL2NVRAM.symmetricKey$
BY $\langle 2 \rangle 1$ DEF $LL2Refinement$

$\langle 4 \rangle 2.$ $ll1sResult.newPrivateState = ll2SResult.newPrivateState$
BY $\langle 3 \rangle 8$

$\langle 4 \rangle 3.$ QED

BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$ DEF *ll1NewPrivateStateEnc*, *ll2NewPrivateStateEnc*

There is no definition in the Memoir-Basic spec that corresponds to *ll2CurrentHistorySummary* in the Memoir-Opt spec. Instead, the corresponding value is the history summary field of the *NVRAM* variable.

$\langle 3 \rangle 10.$ *HistorySummariesMatch*(
LL1NVRAM.historySummary,
ll2CurrentHistorySummary,
LL2NVRAM.hashBarrier′)

The history summary in the Memoir-Basic *NVRAM* matches the logical history summary in the Memoir-Opt *NVRAM* and *SPCR*, as specified by the refinement.

$\langle 4 \rangle 1.$ *HistorySummariesMatch*(
LL1NVRAM.historySummary,
LL2NVRAMLogicalHistorySummary,
LL2NVRAM.hashBarrier)

BY $\langle 2 \rangle 1$ DEF *LL2Refinement*

In the Memoir-Opt spec, the logical history summary in the *NVRAM* and *SPCR* equals the *ll2CurrentHistorySummary* defined by the *LL2RepeatOperation* action. This is a fairly straightforward equivalence by definition, but it requires separate consideration for the two cases of whether an extension is in progress.

$\langle 4 \rangle 2.$ *LL2NVRAMLogicalHistorySummary* = *ll2CurrentHistorySummary*

$\langle 5 \rangle 1.$ CASE *LL2NVRAM.extensionInProgress*

$\langle 6 \rangle 1.$ *LL2NVRAMLogicalHistorySummary* = [
anchor $\mapsto$ *LL2NVRAM.historySummaryAnchor*,
extension $\mapsto$ *LL2SPCR*]

$\langle 7 \rangle 1.$ *LL2SPCR* $\neq$ *BaseHashValue*

BY $\langle 3 \rangle 2$, $\langle 5 \rangle 1$

$\langle 7 \rangle 2.$ QED

BY $\langle 5 \rangle 1$, $\langle 7 \rangle 1$ DEF *LL2NVRAMLogicalHistorySummary*

$\langle 6 \rangle 2.$ *ll2CurrentHistorySummary* = [
anchor $\mapsto$ *LL2NVRAM.historySummaryAnchor*,
extension $\mapsto$ *LL2SPCR*]

BY DEF *ll2CurrentHistorySummary*

$\langle 6 \rangle 3.$ QED

BY $\langle 6 \rangle 1$, $\langle 6 \rangle 2$

$\langle 5 \rangle 2.$ CASE $\neg$*LL2NVRAM.extensionInProgress*

$\langle 6 \rangle 1.$ *LL2NVRAMLogicalHistorySummary* = [
anchor $\mapsto$ *LL2NVRAM.historySummaryAnchor*,
extension $\mapsto$ *BaseHashValue*]

BY $\langle 5 \rangle 2$ DEF *LL2NVRAMLogicalHistorySummary*

$\langle 6 \rangle 2.$ *ll2CurrentHistorySummary* = [
anchor $\mapsto$ *LL2NVRAM.historySummaryAnchor*,
extension $\mapsto$ *LL2SPCR*]

BY DEF *ll2CurrentHistorySummary*

$\langle 6 \rangle 3.$ *LL2SPCR* = *BaseHashValue*

BY $\langle 3 \rangle 2$, $\langle 5 \rangle 2$

$\langle 6 \rangle 4.$ QED

BY $\langle 6 \rangle 1$, $\langle 6 \rangle 2$, $\langle 6 \rangle 3$

$\langle 5 \rangle 3.$ QED

BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$

$\langle 4 \rangle 3.$ UNCHANGED *LL2NVRAM.hashBarrier*

BY $\langle 3 \rangle 2$

$\langle 4 \rangle 4.$ QED

BY $\langle 4\rangle 1$, $\langle 4\rangle 2$, $\langle 4\rangle 3$

The new state hashes are directly equal across the two specs.

$\langle 3\rangle 11.$ $ll1NewStateHash = ll2NewStateHash$

$\langle 4\rangle 1.$ $ll1sResult.newPublicState = ll2SResult.newPublicState$

BY $\langle 3\rangle 8$

$\langle 4\rangle 2.$ $ll1NewPrivateStateEnc = ll2NewPrivateStateEnc$

BY $\langle 3\rangle 9$

$\langle 4\rangle 3.$ QED

BY $\langle 4\rangle 1$, $\langle 4\rangle 2$ DEF $ll1NewStateHash$, $ll2NewStateHash$

The new authenticators match across the two specs, as specified by the *AuthenticatorsMatch* predicate.

$\langle 3\rangle 12.$ $AuthenticatorsMatch($
$ll1NewAuthenticator,$
$ll2NewAuthenticator,$
$LL2NVRAM.symmetricKey',$
$LL2NVRAM.hashBarrier')$

First, we prove some types needed by the definition of the *AuthenticatorsMatch* predicate.

$\langle 4\rangle 1.$ $ll2NewStateHash \in HashType$

BY $\langle 3\rangle 3$

$\langle 4\rangle 2.$ $LL1NVRAM.historySummary \in HashType$

BY $\langle 2\rangle 1$ DEF $LL2Refinement$, $LL1TrustedStorageType$

$\langle 4\rangle 3.$ $ll2CurrentHistorySummary \in HistorySummaryType$

BY $\langle 3\rangle 3$

We then prove that, in the Memoir-Opt spec, the new authenticator is a valid *MAC* for the new history state binding. We will use the *MACComplete* property.

$\langle 4\rangle 4.$ $ValidateMAC(LL2NVRAM.symmetricKey', ll2NewHistoryStateBinding, ll2NewAuthenticator)$

In the Memoir-Opt spec, the new authenticator is generated as a *MAC* of the new history state binding.

$\langle 5\rangle 1.$ $ll2NewAuthenticator =$
$GenerateMAC(LL2NVRAM.symmetricKey', ll2NewHistoryStateBinding)$

$\langle 6\rangle 1.$ UNCHANGED $LL2NVRAM.symmetricKey$

BY $\langle 3\rangle 2$

$\langle 6\rangle 2.$ QED

BY $\langle 6\rangle 1$ DEF $ll2NewAuthenticator$

We can thus use the *MACComplete* property to show that the generated *MAC* validates appropriately. To do this, we first need to prove some types.

$\langle 5\rangle 2.$ $LL2NVRAM.symmetricKey' \in SymmetricKeyType$

BY $\langle 2\rangle 1$, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$

$\langle 5\rangle 3.$ $ll2NewHistoryStateBinding \in HashType$

BY $\langle 3\rangle 3$

Then, we appeal to the *MACComplete* property in a straightforward way.

$\langle 5\rangle 4.$ QED

BY $\langle 5\rangle 1$, $\langle 5\rangle 2$, $\langle 5\rangle 3$, $MACComplete$

We then prove that, in the Memoir-Basic spec, the new authenticator is generated as a *MAC* of the new history state binding.

$\langle 4\rangle 5.$ $ll1NewAuthenticator = GenerateMAC(LL2NVRAM.symmetricKey', ll1NewHistoryStateBinding)$

$\langle 5\rangle 1.$ $LL1NVRAM.symmetricKey = LL2NVRAM.symmetricKey'$

$\langle 6\rangle 1.$ $LL1NVRAM.symmetricKey = LL2NVRAM.symmetricKey$

BY $\langle 2\rangle 1$ DEF $LL2Refinement$

$\langle 6\rangle 2.$ UNCHANGED $LL2NVRAM.symmetricKey$

BY $\langle 3\rangle 2$

$\langle 6\rangle 3.$ QED

BY ⟨6⟩1, ⟨6⟩2

⟨5⟩2. QED

BY ⟨5⟩1 DEF *ll1NewAuthenticator*

The new history summaries match across the two specs, as we proved above.

⟨4⟩6. *HistorySummariesMatch*(
LL1NVRAM.historySummary, *ll2CurrentHistorySummary*, *LL2NVRAM.hashBarrier′*)

BY ⟨3⟩10

We then invoke the definition of the *AuthenticatorsMatch* predicate.

⟨4⟩7. QED

BY ⟨3⟩11, ⟨4⟩1, ⟨4⟩2, ⟨4⟩3, ⟨4⟩4, ⟨4⟩5, ⟨4⟩6

DEF *AuthenticatorsMatch*, *ll1NewHistoryStateBinding*, *ll2NewHistoryStateBinding*, *ll2CurrentHistorySummary*, *ll2CurrentHistorySummaryHash*

The remainder of the proof for *LL2RepeatOperation* is a series of assertions, one for each conjunct in the definition of the *LL1RepeatOperation* action.

The first conjunct in *LL1RepeatOperation*. This is basically just an application of *AuthenticatorValidatedLemma*

⟨3⟩13. *ValidateMAC*(*LL1NVRAM.symmetricKey*, *ll1HistoryStateBinding*, *LL1RAM.authenticator*)

We need the fact that the symmetric keys in the *NVRAM* are equal across the two specs.

⟨4⟩1. *LL1NVRAM.symmetricKey* = *LL2NVRAM.symmetricKey*

BY ⟨2⟩1 DEF *LL2Refinement*

We prove the types that are needed for *AuthenticatorValidatedLemma*.

⟨4⟩2. *ll2StateHash* ∈ *HashType*

BY ⟨3⟩3

⟨4⟩3. *LL1RAM.historySummary* ∈ *HashType*

BY ⟨2⟩1 DEF *LL2Refinement*, *LL1UntrustedStorageType*

⟨4⟩4. *LL2RAM.historySummary* ∈ *HistorySummaryType*

BY ⟨2⟩1, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*

⟨4⟩5. *LL1RAM.authenticator* ∈ *MACType*

BY ⟨2⟩1 DEF *LL2Refinement*, *LL1UntrustedStorageType*

⟨4⟩6. *LL2RAM.authenticator* ∈ *MACType*

BY ⟨2⟩1, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*

⟨4⟩7. *LL2NVRAM.symmetricKey* ∈ *SymmetricKeyType*

BY ⟨2⟩1, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*

⟨4⟩8. *LL2NVRAM.hashBarrier* ∈ *HashType*

BY ⟨2⟩1, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*

There are three preconditions for *AuthenticatorValidatedLemma*. The first precondition follows from the refinement.

⟨4⟩9. *HistorySummariesMatch*(
LL1RAM.historySummary, *LL2RAM.historySummary*, *LL2NVRAM.hashBarrier*)

BY ⟨2⟩1 DEF *LL2Refinement*

The second precondition also follows from the refinement.

⟨4⟩10. PICK *symmetricKey* ∈ *SymmetricKeyType* :
AuthenticatorsMatch(
LL1RAM.authenticator,
LL2RAM.authenticator,
symmetricKey,
LL2NVRAM.hashBarrier)

BY ⟨2⟩1 DEF *LL2Refinement*

The third precondition follows from the definition of the *LL2RepeatOperation* action.

⟨4⟩11. *ValidateMAC*(*LL2NVRAM.symmetricKey*, *ll2HistoryStateBinding*, *LL2RAM.authenticator*)

BY ⟨3⟩2 DEF *ll2HistoryStateBinding*, *ll2StateHash*, *ll2HistorySummaryHash*

⟨4⟩12. QED

Ideally, this QED step should just read:

BY ⟨3⟩6, ⟨4⟩1, ⟨4⟩2, ⟨4⟩3, ⟨4⟩4, ⟨4⟩5, ⟨4⟩6, ⟨4⟩7, ⟨4⟩8, ⟨4⟩9, ⟨4⟩10, ⟨4⟩11,
AuthenticatorValidatedLemma
DEF *ll1HistoryStateBinding*, *ll2HistoryStateBinding*, *ll2HistorySummaryHash*

However, the prover seems to get a little confused in this instance. We make life easier for the prover by explicitly staging the instantiation of the quantified variables within the definition of *AuthenticatorValidatedLemma*.

⟨5⟩1. $\forall$ *samLL2Authenticator* $\in$ *MACType*,
samSymmetricKey2 $\in$ *SymmetricKeyType*,
samHashBarrier $\in$ *HashType* :
AuthenticatorValidatedLemma!(*ll2StateHash*, *LL1RAM.historySummary*,
LL2RAM.historySummary, *LL1RAM.authenticator*, *samLL2Authenticator*,
symmetricKey, *samSymmetricKey2*, *samHashBarrier*)!1
BY ⟨4⟩2, ⟨4⟩3, ⟨4⟩4, ⟨4⟩5, *AuthenticatorValidatedLemma*

⟨5⟩2. *AuthenticatorValidatedLemma*!(*ll2StateHash*, *LL1RAM.historySummary*,
LL2RAM.historySummary, *LL1RAM.authenticator*, *LL2RAM.authenticator*,
symmetricKey, *LL2NVRAM.symmetricKey*, *LL2NVRAM.hashBarrier*)!1
BY ⟨4⟩2, ⟨4⟩3, ⟨4⟩4, ⟨4⟩5, ⟨4⟩6, ⟨4⟩7, ⟨4⟩8, ⟨4⟩10, ⟨5⟩1

⟨5⟩3. QED
BY ⟨3⟩6, ⟨4⟩1, ⟨4⟩2, ⟨4⟩3, ⟨4⟩4, ⟨4⟩5, ⟨4⟩6, ⟨4⟩7, ⟨4⟩8, ⟨4⟩9, ⟨4⟩10, ⟨4⟩11, ⟨5⟩2
DEF *ll1HistoryStateBinding*, *ll2HistoryStateBinding*, *ll2HistorySummaryHash*

The second conjunct in *LL1RepeatOperation*: In the Memoir-Basic spec, the history summary in the *NVRAM* equals the hash of the history summary in the RAM and the input.

⟨3⟩14. *LL1NVRAM.historySummary* $=$ *Hash*(*LL1RAM.historySummary*, *input*)

We separately tackle the two cases of whether a checkpoint was taken before the input was processed.

⟨4⟩1. $\lor$ *ll2CurrentHistorySummary* $=$ *ll2CheckpointedNewHistorySummary*
$\lor$ *ll2CurrentHistorySummary* $=$ *ll2CheckpointedNewCheckpointedHistorySummary*
BY ⟨3⟩2
DEF *ll2CurrentHistorySummary*, *ll2NewHistorySummary*,
ll2CheckpointedHistorySummary, *ll2NewCheckpointedHistorySummary*,
ll2CheckpointedNewHistorySummary, *ll2CheckpointedNewCheckpointedHistorySummary*

Before proceeding to the cases, we'll prove one type fact that will be needed several places below.

⟨4⟩2. *Hash*(*LL1RAM.historySummary*, *input*) $\in$ *HashType*
⟨5⟩1. *LL1RAM.historySummary* $\in$ *HashDomain*
⟨6⟩1. *LL1RAM.historySummary* $\in$ *HashType*
BY ⟨2⟩1 DEF *LL2Refinement*, *LL1UntrustedStorageType*
⟨6⟩2. QED
BY ⟨6⟩1 DEF *HashDomain*
⟨5⟩2. *input* $\in$ *HashDomain*
⟨6⟩1. *input* $\in$ *InputType*
⟨7⟩1. *input* $\in$ *LL2AvailableInputs*
BY ⟨3⟩2
⟨7⟩2. *LL2AvailableInputs* $\subseteq$ *InputType*
BY ⟨2⟩1 DEF *LL2TypeInvariant*
⟨7⟩3. QED
BY ⟨7⟩1, ⟨7⟩2
⟨6⟩2. QED
BY ⟨6⟩1 DEF *HashDomain*
⟨5⟩3. QED
BY ⟨5⟩1, ⟨5⟩2, *HashTypeSafe*

In the first case, a checkpoint was not taken before the input was processed.

$\langle 4\rangle 3.$ CASE $ll2CurrentHistorySummary = ll2CheckpointedNewHistorySummary$

We will be employing the *HistorySummariesMatchAcrossCheckpointLemma*, which has three preconditions. The first is that the history summaries in the *NVRAM* match across the two specs, which is true by refinement.

$\langle 5\rangle 1.$ $HistorySummariesMatch($
$LL1NVRAM.historySummary,$
$LL2NVRAMLogicalHistorySummary,$
$LL2NVRAM.hashBarrier)$
BY $\langle 2\rangle 1$ DEF $LL2Refinement$

The second precondition is that the Memoir-Basic history summary formed from the hash of the history summary in the RAM and the input matches the new history summary in the Memoir-Opt spec.

$\langle 5\rangle 2.$ $HistorySummariesMatch($
$Hash(LL1RAM.historySummary, input),$
$ll2NewHistorySummary,$
$LL2NVRAM.hashBarrier)$

First, we prove that the *HistorySummariesMatch* predicate equals the *HistorySummariesMatchRecursion* predicate in this case. We assert each condition required by the definition of the predicate.

$\langle 6\rangle 1.$ $HistorySummariesMatch($
$Hash(LL1RAM.historySummary, input),$
$ll2NewHistorySummary,$
$LL2NVRAM.hashBarrier) =$
$HistorySummariesMatchRecursion($
$Hash(LL1RAM.historySummary, input),$
$ll2NewHistorySummary,$
$LL2NVRAM.hashBarrier)$

We begin by proving the types for the *HistorySummariesMatchRecursion* predicate.

$\langle 7\rangle 1.$ $Hash(LL1RAM.historySummary, input) \in HashType$
BY $\langle 4\rangle 2$
$\langle 7\rangle 2.$ $ll2NewHistorySummary \in HistorySummaryType$
BY $\langle 3\rangle 3$
$\langle 7\rangle 3.$ $LL2NVRAM.hashBarrier \in HashType$
BY $\langle 2\rangle 1$, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$
$\langle 7\rangle$ $ll2InitialHistorySummary \triangleq [anchor \mapsto BaseHashValue, extension \mapsto BaseHashValue]$

We then prove that this is not the base case for the *HistorySummariesMatch* predicate.

$\langle 7\rangle 4.$ $ll2NewHistorySummary \neq ll2InitialHistorySummary$

This proof has a lot of sub-steps, but it is pretty simple. We just use the *BaseHashValueUnique* property to show that the extension field in the *ll2NewHistorySummary* record cannot match the base hash value, which is the value of the extension field in the initial history summary.

$\langle 8\rangle 1.$ $ll2NewHistorySummary =$
$Successor(LL2RAM.historySummary, input, LL2NVRAM.hashBarrier)$
BY DEF $ll2NewHistorySummary$
$\langle 8\rangle 2.$ $ll2NewHistorySummary.extension =$
$Hash(LL2RAM.historySummary.extension, Hash(LL2NVRAM.hashBarrier, input))$
BY $\langle 8\rangle 1$ DEF $Successor$
$\langle 8\rangle 3.$ $ll2NewHistorySummary.extension \neq BaseHashValue$
$\langle 9\rangle 1.$ $LL2RAM.historySummary.extension \in HashDomain$
$\langle 10\rangle 1.$ $LL2RAM.historySummary.extension \in HashType$
$\langle 11\rangle 1.$ $LL2RAM.historySummary \in HistorySummaryType$
BY $\langle 2\rangle 1$, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$
$\langle 11\rangle 2.$ QED
BY $\langle 11\rangle 1$ DEF $HistorySummaryType$
$\langle 10\rangle 2.$ QED

BY $\langle 10\rangle 1$ DEF *HashDomain*
$\langle 9\rangle 2$. $Hash(LL2NVRAM.hashBarrier, input) \in HashDomain$
$\langle 10\rangle 1$. $Hash(LL2NVRAM.hashBarrier, input) \in HashType$
$\langle 11\rangle 1$. $LL2NVRAM.hashBarrier \in HashDomain$
$\langle 12\rangle 1$. $LL2NVRAM.hashBarrier \in HashType$
BY $\langle 2\rangle 1$, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*
$\langle 12\rangle 2$. QED
BY $\langle 12\rangle 1$ DEF *HashDomain*
$\langle 11\rangle 2$. $input \in HashDomain$
$\langle 12\rangle 1$. $input \in InputType$
$\langle 13\rangle 1$. $input \in LL2AvailableInputs$
BY $\langle 3\rangle 2$
$\langle 13\rangle 2$. $LL2AvailableInputs \subseteq InputType$
BY $\langle 2\rangle 1$ DEF *LL2TypeInvariant*
$\langle 13\rangle 3$. QED
BY $\langle 13\rangle 1$, $\langle 13\rangle 2$
$\langle 12\rangle 2$. QED
BY $\langle 12\rangle 1$ DEF *HashDomain*
$\langle 11\rangle 3$. QED
BY $\langle 11\rangle 1$, $\langle 11\rangle 2$, *HashTypeSafe*
$\langle 10\rangle 2$. QED
BY $\langle 10\rangle 1$ DEF *HashDomain*
$\langle 9\rangle 3$. QED
BY $\langle 8\rangle 2$, $\langle 9\rangle 1$, $\langle 9\rangle 2$, *BaseHashValueUnique*
$\langle 8\rangle 4$. QED
BY $\langle 8\rangle 3$

Since this is not the base case, the *HistorySummariesMatch* predicate equals the *HistorySummariesMatchRecursion* predicate.

$\langle 7\rangle 5$. QED
BY $\langle 7\rangle 1$, $\langle 7\rangle 2$, $\langle 7\rangle 3$, $\langle 7\rangle 4$, *HistorySummariesMatchDefinition*

Then, we prove that the *HistorySummariesMatchRecursion* predicate is satisfied. We assert each condition required by the definition of the predicate.

$\langle 6\rangle 2$. $HistorySummariesMatchRecursion($
$Hash(LL1RAM.historySummary, input),$
$ll2NewHistorySummary,$
$LL2NVRAM.hashBarrier)$

We begin by proving the types for the existentially quantified variables in the *HistorySummariesMatchRecursion* predicate.

$\langle 7\rangle 1$. $input \in InputType$
$\langle 8\rangle 1$. $input \in LL2AvailableInputs$
BY $\langle 3\rangle 2$
$\langle 8\rangle 2$. $LL2AvailableInputs \subseteq InputType$
BY $\langle 2\rangle 1$ DEF *LL2TypeInvariant*
$\langle 8\rangle 3$. QED
BY $\langle 8\rangle 1$, $\langle 8\rangle 2$
$\langle 7\rangle 2$. $LL1RAM.historySummary \in HashType$
BY $\langle 2\rangle 1$ DEF *LL2Refinement*, *LL1UntrustedStorageType*
$\langle 7\rangle 3$. $LL2RAM.historySummary \in HistorySummaryType$
BY $\langle 2\rangle 1$, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*

We then prove the three conjuncts in the *HistorySummariesMatchRecursion* predicate. The first conjunct follows directly from the refinement.

$\langle 7\rangle 4$. $HistorySummariesMatch($

$LL1RAM.historySummary$,
$LL2RAM.historySummary$,
$LL2NVRAM.hashBarrier$)
BY ⟨2⟩1 DEF *LL2Refinement*

The second conjunct in the *HistorySummariesMatchRecursion* predicate is that $Hash(LL1RAM.historySummary, input)$ is equal to itself. We do not bother writing this.

The third conjunct in the *HistorySummariesMatchRecursion* predicate is true by definition.

⟨7⟩6. $LL2HistorySummaryIsSuccessor($
$ll2NewHistorySummary, LL2RAM.historySummary, input, LL2NVRAM.hashBarrier)$
BY DEF *LL2HistorySummaryIsSuccessor*, *ll2NewHistorySummary*

All conjuncts in the *HistorySummariesMatchRecursion* predicate are satisfied.

⟨7⟩7. QED
BY ⟨7⟩1, ⟨7⟩2, ⟨7⟩3, ⟨7⟩4, ⟨7⟩6 DEF *HistorySummariesMatchRecursion*

Since the *HistorySummariesMatch* predicate equals the *HistorySummariesMatchRecursion* predicate, and the latter predicate is satisfied, the former predicate is satisfied.

⟨6⟩3. QED
BY ⟨6⟩1, ⟨6⟩2

The third precondition is that in the Memoir-Opt spec, the logical history summary in the *NVRAM* and *SPCR* equals the checkpointed new history summary. This follows from a straightforward expansion of definitions, given that an extension is not in progress when an *LL2RepeatOperation* action is executed.

⟨5⟩3. $LL2NVRAMLogicalHistorySummary = ll2CheckpointedNewHistorySummary$
⟨6⟩1. $\neg LL2NVRAM.extensionInProgress$
BY ⟨3⟩2
⟨6⟩2. $LL2NVRAMLogicalHistorySummary = ll2CurrentHistorySummary$
⟨7⟩1. $LL2NVRAMLogicalHistorySummary = [$
$anchor \mapsto LL2NVRAM.historySummaryAnchor$,
$extension \mapsto BaseHashValue]$
BY ⟨6⟩1 DEF *LL2NVRAMLogicalHistorySummary*
⟨7⟩2. $ll2CurrentHistorySummary = [$
$anchor \mapsto LL2NVRAM.historySummaryAnchor$,
$extension \mapsto LL2SPCR]$
BY DEF *ll2CurrentHistorySummary*
⟨7⟩3. $LL2SPCR = BaseHashValue$
BY ⟨3⟩2, ⟨6⟩1
⟨7⟩4. QED
BY ⟨7⟩1, ⟨7⟩2, ⟨7⟩3
⟨6⟩3. $ll2CurrentHistorySummary = ll2CheckpointedNewHistorySummary$
BY ⟨4⟩3
⟨6⟩4. QED
BY ⟨6⟩2, ⟨6⟩3

We use the *HistorySummariesMatchAcrossCheckpointLemma* to prove that the fields are equal. This requires proving some types.

⟨5⟩4. QED
⟨6⟩1. $LL1NVRAM.historySummary \in HashType$
BY ⟨2⟩1 DEF *LL2Refinement*, *LL1TrustedStorageType*
⟨6⟩2. $Hash(LL1RAM.historySummary, input) \in HashType$
BY ⟨4⟩2
⟨6⟩3. $LL2NVRAMLogicalHistorySummary \in HistorySummaryType$
BY ⟨2⟩1, *LL2NVRAMLogicalHistorySummaryTypeSafe*
⟨6⟩4. $ll2NewHistorySummary \in HistorySummaryType$
BY ⟨3⟩3

$\langle 6\rangle 5.$ $LL2NVRAM.hashBarrier \in HashType$
BY $\langle 2\rangle 1$, $LL2SubtypeImplicationLemma$ DEF $LL2SubtypeImplication$
$\langle 6\rangle 6.$ QED
BY $\langle 5\rangle 1$, $\langle 5\rangle 2$, $\langle 5\rangle 3$, $\langle 6\rangle 1$, $\langle 6\rangle 2$, $\langle 6\rangle 3$, $\langle 6\rangle 4$, $\langle 6\rangle 5$,
$HistorySummariesMatchAcrossCheckpointLemma$
DEF $ll2CheckpointedNewHistorySummary$

In the second case, a checkpoint was taken before the input was processed.

$\langle 4\rangle 4.$ CASE $ll2CurrentHistorySummary = ll2CheckpointedNewCheckpointedHistorySummary$

We will be employing the *HistorySummariesMatchAcrossCheckpointLemma*, which has three preconditions. The first is that the history summaries in the *NVRAM* match across the two specs, which is true by refinement.

$\langle 5\rangle 1.$ $HistorySummariesMatch($
$LL1NVRAM.historySummary,$
$LL2NVRAMLogicalHistorySummary,$
$LL2NVRAM.hashBarrier)$
BY $\langle 2\rangle 1$ DEF $LL2Refinement$

The second precondition is that the Memoir-Basic history summary formed from the hash of the history summary in the RAM and the input matches the new checkpointed history summary in the Memoir-Opt spec.

$\langle 5\rangle 2.$ $HistorySummariesMatch($
$Hash(LL1RAM.historySummary, input),$
$ll2NewCheckpointedHistorySummary,$
$LL2NVRAM.hashBarrier)$

First, we prove that the *HistorySummariesMatch* predicate equals the *HistorySummariesMatchRecursion* predicate in this case. We assert each condition required by the definition of the predicate.

$\langle 6\rangle 1.$ $HistorySummariesMatch($
$Hash(LL1RAM.historySummary, input),$
$ll2NewCheckpointedHistorySummary,$
$LL2NVRAM.hashBarrier) =$
$HistorySummariesMatchRecursion($
$Hash(LL1RAM.historySummary, input),$
$ll2NewCheckpointedHistorySummary,$
$LL2NVRAM.hashBarrier)$

We begin by proving the types for the *HistorySummariesMatchRecursion* predicate.

$\langle 7\rangle 1.$ $Hash(LL1RAM.historySummary, input) \in HashType$
BY $\langle 4\rangle 2$
$\langle 7\rangle 2.$ $ll2NewCheckpointedHistorySummary \in HistorySummaryType$
BY $\langle 3\rangle 3$
$\langle 7\rangle 3.$ $LL2NVRAM.hashBarrier \in HashType$
BY $\langle 2\rangle 1$, $LL2SubtypeImplicationLemma$ DEF $LL2SubtypeImplication$
$\langle 7\rangle$ $ll2InitialHistorySummary \triangleq [anchor \mapsto BaseHashValue, extension \mapsto BaseHashValue]$

We then prove that this is not the base case for the *HistorySummariesMatch* predicate.

$\langle 7\rangle 4.$ $ll2NewCheckpointedHistorySummary \neq ll2InitialHistorySummary$

This proof has a lot of sub-steps, but it is pretty simple. We just use the *BaseHashValueUnique* property to show that the extension field in the *ll2NewCheckpointedHistorySummary* record cannot match the base hash value, which is the value of the extension field in the initial history summary.

$\langle 8\rangle 1.$ $ll2NewCheckpointedHistorySummary =$
$Successor(ll2CheckpointedHistorySummary, input, LL2NVRAM.hashBarrier)$
BY DEF $ll2NewCheckpointedHistorySummary$
$\langle 8\rangle 2.$ $ll2NewCheckpointedHistorySummary.extension =$
$Hash(ll2CheckpointedHistorySummary.extension,$
$Hash(LL2NVRAM.hashBarrier, input))$

BY ⟨8⟩1 DEF *Successor*

⟨8⟩3. $ll2NewCheckpointedHistorySummary.extension \neq BaseHashValue$

⟨9⟩1. $ll2CheckpointedHistorySummary.extension \in HashDomain$

⟨10⟩1. $ll2CheckpointedHistorySummary.extension \in HashType$

⟨11⟩1. $ll2CheckpointedHistorySummary \in HistorySummaryType$

BY ⟨3⟩3

⟨11⟩2. QED

BY ⟨11⟩1 DEF *HistorySummaryType*

⟨10⟩2. QED

BY ⟨10⟩1 DEF *HashDomain*

⟨9⟩2. $Hash(LL2NVRAM.hashBarrier, input) \in HashDomain$

⟨10⟩1. $Hash(LL2NVRAM.hashBarrier, input) \in HashType$

⟨11⟩1. $LL2NVRAM.hashBarrier \in HashDomain$

⟨12⟩1. $LL2NVRAM.hashBarrier \in HashType$

BY ⟨2⟩1, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*

⟨12⟩2. QED

BY ⟨12⟩1 DEF *HashDomain*

⟨11⟩2. $input \in HashDomain$

⟨12⟩1. $input \in InputType$

⟨13⟩1. $input \in LL2AvailableInputs$

BY ⟨3⟩2

⟨13⟩2. $LL2AvailableInputs \subseteq InputType$

BY ⟨2⟩1 DEF *LL2TypeInvariant*

⟨13⟩3. QED

BY ⟨13⟩1, ⟨13⟩2

⟨12⟩2. QED

BY ⟨12⟩1 DEF *HashDomain*

⟨11⟩3. QED

BY ⟨11⟩1, ⟨11⟩2, *HashTypeSafe*

⟨10⟩2. QED

BY ⟨10⟩1 DEF *HashDomain*

⟨9⟩3. QED

BY ⟨8⟩2, ⟨9⟩1, ⟨9⟩2, *BaseHashValueUnique*

⟨8⟩4. QED

BY ⟨8⟩3

Since this is not the base case, the *HistorySummariesMatch* predicate equals the *HistorySummariesMatchRecursion* predicate.

⟨7⟩5. QED

BY ⟨7⟩1, ⟨7⟩2, ⟨7⟩3, ⟨7⟩4, *HistorySummariesMatchDefinition*

Then, we prove that the *HistorySummariesMatchRecursion* predicate is satisfied. We assert each condition required by the definition of the predicate.

⟨6⟩2. $HistorySummariesMatchRecursion($
$Hash(LL1RAM.historySummary, input),$
$ll2NewCheckpointedHistorySummary,$
$LL2NVRAM.hashBarrier)$

We begin by proving the types for the existentially quantified variables in the *HistorySummariesMatchRecursion* predicate.

⟨7⟩1. $input \in InputType$

⟨8⟩1. $input \in LL2AvailableInputs$

BY ⟨3⟩2

⟨8⟩2. $LL2AvailableInputs \subseteq InputType$

BY ⟨2⟩1 DEF *LL2TypeInvariant*

⟨8⟩3. QED
BY ⟨8⟩1, ⟨8⟩2
⟨7⟩2. $LL1RAM.historySummary \in HashType$
BY ⟨2⟩1 DEF $LL2Refinement$, $LL1UntrustedStorageType$
⟨7⟩3. $Checkpoint(LL2RAM.historySummary) \in HistorySummaryType$
⟨8⟩1. $LL2RAM.historySummary \in HistorySummaryType$
BY ⟨2⟩1, $LL2SubtypeImplicationLemma$ DEF $LL2SubtypeImplication$
⟨8⟩2. QED
BY ⟨8⟩1, $CheckpointTypeSafe$

We then prove the three conjuncts in the *HistorySummariesMatchRecursion* predicate.

The first conjunct is a recursive instance of the *HistorySummariesMatch* predicate, namely that the history summary in the Memoir-Basic RAM matches the checkpointed history summary in the Memoir-Opt RAM. We have to recursively expand the definition.

⟨7⟩4. $HistorySummariesMatch($
$LL1RAM.historySummary,$
$Checkpoint(LL2RAM.historySummary),$
$LL2NVRAM.hashBarrier)$

From the refinement, we know that the history summary in the Memoir-Basic RAM matches the uncheckpointed history summary in the Memoir-Opt RAM.

⟨8⟩1. $HistorySummariesMatch($
$LL1RAM.historySummary,$
$LL2RAM.historySummary,$
$LL2NVRAM.hashBarrier)$
BY ⟨2⟩1 DEF $LL2Refinement$

We separately consider the base case and recursive case. We will apply the *HistorySummariesMatchDefinition* in both cases, so we prove the necessary types once up front.

⟨8⟩2. $LL1RAM.historySummary \in HashType$
BY ⟨2⟩1 DEF $LL2Refinement$, $LL1UntrustedStorageType$
⟨8⟩3. $Checkpoint(LL2RAM.historySummary) \in HistorySummaryType$
BY ⟨7⟩3
⟨8⟩4. $LL2NVRAM.hashBarrier \in HashType$
BY ⟨2⟩1, $LL2SubtypeImplicationLemma$ DEF $LL2SubtypeImplication$
⟨8⟩ $ll2InitialHistorySummary \triangleq [anchor \mapsto BaseHashValue, extension \mapsto BaseHashValue]$

The base case is very simple.

⟨8⟩5. CASE $LL2RAM.historySummary = ll2InitialHistorySummary$
⟨9⟩1. $Checkpoint(LL2RAM.historySummary) = ll2InitialHistorySummary$
BY ⟨8⟩5 DEF $Checkpoint$
⟨9⟩2. $LL1RAM.historySummary = BaseHashValue$
⟨10⟩1. $LL2RAM.historySummary \in HistorySummaryType$
BY ⟨2⟩1, $LL2SubtypeImplicationLemma$ DEF $LL2SubtypeImplication$
⟨10⟩2. QED
BY ⟨8⟩1, ⟨8⟩2, ⟨8⟩4, ⟨8⟩5, ⟨10⟩1, $HistorySummariesMatchDefinition$
⟨9⟩3. QED
BY ⟨8⟩2, ⟨8⟩3, ⟨8⟩4, ⟨9⟩1, ⟨9⟩2, $HistorySummariesMatchDefinition$

The recursive case is pretty involved.

⟨8⟩6. CASE $LL2RAM.historySummary \neq ll2InitialHistorySummary$

First, we prove that the *HistorySummariesMatch* predicate equals the *HistorySummariesMatchRecursion* predicate in this case.

⟨9⟩1. $HistorySummariesMatch($
$LL1RAM.historySummary,$

$\quad\quad Checkpoint(LL2RAM.historySummary),$
$\quad\quad LL2NVRAM.hashBarrier) =$
$\quad HistorySummariesMatchRecursion($
$\quad\quad LL1RAM.historySummary,$
$\quad\quad Checkpoint(LL2RAM.historySummary),$
$\quad\quad LL2NVRAM.hashBarrier)$

There is only one sub-step, which is showing that the Memoir-Opt checkpointed history summary is not equal to the initial history summary. This follows naturally from the case we're in, but it takes a lot of tedious steps to get the prover to recognize this.

$\langle 10\rangle 1.$ $Checkpoint(LL2RAM.historySummary) \neq ll2InitialHistorySummary$
$\langle 11\rangle 1.$ CASE $LL2RAM.historySummary.extension = BaseHashValue$
$\langle 12\rangle 1.$ $Checkpoint(LL2RAM.historySummary) = LL2RAM.historySummary$
BY $\langle 11\rangle 1$ DEF $Checkpoint$
$\langle 12\rangle 2.$ QED
BY $\langle 8\rangle 6$, $\langle 11\rangle 1$, $\langle 12\rangle 1$
$\langle 11\rangle 2.$ CASE $LL2RAM.historySummary.extension \neq BaseHashValue$
$\langle 12\rangle 1.$ $Checkpoint(LL2RAM.historySummary) = [$
$\quad anchor \mapsto Hash($
$\quad\quad LL2RAM.historySummary.anchor,$
$\quad\quad LL2RAM.historySummary.extension),$
$\quad extension \mapsto BaseHashValue]$
BY $\langle 11\rangle 2$ DEF $Checkpoint$
$\langle 12\rangle 2.$ $Checkpoint(LL2RAM.historySummary).anchor = Hash($
$\quad LL2RAM.historySummary.anchor, LL2RAM.historySummary.extension)$
BY $\langle 12\rangle 1$
$\langle 12\rangle 3.$ $Checkpoint(LL2RAM.historySummary).anchor \neq BaseHashValue$
$\langle 13\rangle 1.$ $LL2RAM.historySummary.anchor \in HashDomain$
$\langle 14\rangle 1.$ $LL2RAM.historySummary.anchor \in HashType$
$\langle 15\rangle 1.$ $LL2RAM.historySummary \in HistorySummaryType$
BY $\langle 2\rangle 1$, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$
$\langle 15\rangle 2.$ QED
BY $\langle 15\rangle 1$ DEF $HistorySummaryType$
$\langle 14\rangle 2.$ QED
BY $\langle 14\rangle 1$ DEF $HashDomain$
$\langle 13\rangle 2.$ $LL2RAM.historySummary.extension \in HashDomain$
$\langle 14\rangle 1.$ $LL2RAM.historySummary.extension \in HashType$
$\langle 15\rangle 1.$ $LL2RAM.historySummary \in HistorySummaryType$
BY $\langle 2\rangle 1$, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$
$\langle 15\rangle 2.$ QED
BY $\langle 15\rangle 1$ DEF $HistorySummaryType$
$\langle 14\rangle 2.$ QED
BY $\langle 14\rangle 1$ DEF $HashDomain$
$\langle 13\rangle 3.$ QED
BY $\langle 12\rangle 2$, $\langle 13\rangle 1$, $\langle 13\rangle 2$, $BaseHashValueUnique$
$\langle 12\rangle 4.$ QED
BY $\langle 12\rangle 2$, $\langle 12\rangle 3$
$\langle 11\rangle 3.$ QED
BY $\langle 11\rangle 1$, $\langle 11\rangle 2$
$\langle 10\rangle 2.$ QED
BY $\langle 8\rangle 2$, $\langle 8\rangle 3$, $\langle 8\rangle 4$, $\langle 10\rangle 1$, $HistorySummariesMatchDefinition$

Then, we'll show that the *HistorySummariesMatchRecursion* predicate holds.

$\langle 9\rangle 2.$ $HistorySummariesMatchRecursion($

$LL1RAM.historySummary$,
$Checkpoint(LL2RAM.historySummary)$,
$LL2NVRAM.hashBarrier)$

We'll first pick values for the existential variables inside the *HistorySummariesMatchRecursion* predicate that satisfy the predicate for the uncheckpointed history summary in the Memoir-Opt RAM. We know such variables exist, because this history summary matches the history summary in the Memoir-Basic RAM, by the refinement.

⟨10⟩1. PICK $prevInput \in InputType$,
$previousLL1HistorySummary \in HashType$,
$previousLL2HistorySummary \in HistorySummaryType$:
$HistorySummariesMatchRecursion($
$LL1RAM.historySummary$,
$LL2RAM.historySummary$,
$LL2NVRAM.hashBarrier)!($
$prevInput$,
$previousLL1HistorySummary$,
$previousLL2HistorySummary)$

⟨11⟩1. $LL2RAM.historySummary \in HistorySummaryType$
BY ⟨2⟩1, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*

⟨11⟩2. QED
BY ⟨8⟩1, ⟨8⟩2, ⟨8⟩4, ⟨8⟩6, ⟨11⟩1, *HistorySummariesMatchDefinition*
DEF *HistorySummariesMatchRecursion*

We then assert all of the conditions necessary to satisfy the *HistorySummariesMatchRecursion* for the checkpointed history summary in the Memoir-Opt RAM. The first three of these are identical to the conditions that hold for the uncheckpointed history summary.

⟨10⟩2. $prevInput \in InputType$
BY ⟨10⟩1

⟨10⟩3. $HistorySummariesMatch($
$previousLL1HistorySummary$,
$previousLL2HistorySummary$,
$LL2NVRAM.hashBarrier)$
BY ⟨10⟩1

⟨10⟩4. $LL1RAM.historySummary = Hash(previousLL1HistorySummary, prevInput)$
BY ⟨10⟩1

The last condition is more involved. We show that the checkpointed history summary is a successor of the previous history summary.

⟨10⟩5. $LL2HistorySummaryIsSuccessor($
$Checkpoint(LL2RAM.historySummary)$,
$previousLL2HistorySummary$,
$prevInput$,
$LL2NVRAM.hashBarrier)$

We note that the uncheckpointed history summary is a successor of the previous history summary.

⟨11⟩1. $LL2HistorySummaryIsSuccessor($
$LL2RAM.historySummary$,
$previousLL2HistorySummary$,
$prevInput$,
$LL2NVRAM.hashBarrier)$
BY ⟨10⟩1

The definition of *LL2HistorySummaryIsSuccessor* tells us that there are two ways that the uncheckpointed history summary could be a successor. We will consider the two cases separately.

⟨11⟩2. $\lor$ $LL2RAM.historySummary =$
$Successor($

$previousLL2HistorySummary$,
$prevInput$,
$LL2NVRAM.hashBarrier$)
$\vee$ $LL2RAM.historySummary =$
$Checkpoint$(
$Successor$(
$previousLL2HistorySummary$,
$prevInput$,
$LL2NVRAM.hashBarrier$))
BY ⟨11⟩1 DEF $LL2HistorySummaryIsSuccessor$

The first case is trivial. If the uncheckpointed history summary is a successor by virtue of the Successor operator, then an application of the Checkpoint operater yields the checkpointed history summary.

⟨11⟩3. CASE $LL2RAM.historySummary =$
$Successor(previousLL2HistorySummary, prevInput, LL2NVRAM.hashBarrier)$
⟨12⟩1. $Checkpoint(LL2RAM.historySummary) =$
$Checkpoint$(
$Successor$(
$previousLL2HistorySummary$,
$prevInput$,
$LL2NVRAM.hashBarrier$))
BY ⟨11⟩3
⟨12⟩2. QED
BY ⟨12⟩1 DEF $LL2HistorySummaryIsSuccessor$

The second case is only slightly more involved. If the uncheckpointed history summary is a successor by virtue of the *Successor* operator and the *Checkpoint* operator, then a second application of the *Checkpoint* operater is idempotent.

⟨11⟩4. CASE $LL2RAM.historySummary =$
$Checkpoint$(
$Successor$(
$previousLL2HistorySummary$,
$prevInput$,
$LL2NVRAM.hashBarrier$))
⟨12⟩1. $Checkpoint(LL2RAM.historySummary) =$
$Checkpoint(Checkpoint($
$Successor$(
$previousLL2HistorySummary$,
$prevInput$,
$LL2NVRAM.hashBarrier$)))
BY ⟨11⟩4
⟨12⟩2. $Checkpoint(LL2RAM.historySummary) =$
$Checkpoint$(
$Successor$(
$previousLL2HistorySummary$,
$prevInput$,
$LL2NVRAM.hashBarrier$))
BY ⟨12⟩1 DEF $Checkpoint$
⟨12⟩3. QED
BY ⟨12⟩2 DEF $LL2HistorySummaryIsSuccessor$
⟨11⟩5. QED
BY ⟨11⟩2, ⟨11⟩3, ⟨11⟩4

We have thus shown that the conditions for the *HistorySummariesMatchRecursion* predicate all hold.

⟨10⟩6. QED

BY ⟨8⟩2, ⟨8⟩3, ⟨8⟩4, ⟨10⟩2, ⟨10⟩3, ⟨10⟩4, ⟨10⟩5 DEF *HistorySummariesMatchRecursion*

Since the *HistorySummariesMatch* predicate equals the *HistorySummariesMatchRecursion* predicate, and since the *HistorySummariesMatchRecursion* predicate is satisfied, the *HistorySummariesMatch* predicate is satsisfied.

⟨9⟩3. QED

BY ⟨9⟩1, ⟨9⟩2

Both the base case and the recursive case are satisfied.

⟨8⟩7. QED

BY ⟨8⟩5, ⟨8⟩6

The second conjunct in the *HistorySummariesMatchRecursion* predicate is that $Hash(LL1RAM.historySummary, input)$ is equal to itself. We do not bother writing this.

The third conjunct in the *HistorySummariesMatchRecursion* predicate is true by definition.

⟨7⟩6. $LL2HistorySummaryIsSuccessor($
$ll2NewCheckpointedHistorySummary,$
$Checkpoint(LL2RAM.historySummary),$
$input,$
$LL2NVRAM.hashBarrier)$

BY DEF *LL2HistorySummaryIsSuccessor*, *ll2NewCheckpointedHistorySummary*, *ll2CheckpointedHistorySummary*

All conjuncts in the *HistorySummariesMatchRecursion* predicate are satisfied.

⟨7⟩7. QED

BY ⟨7⟩1, ⟨7⟩2, ⟨7⟩3, ⟨7⟩4, ⟨7⟩6 DEF *HistorySummariesMatchRecursion*

Since the *HistorySummariesMatch* predicate equals the *HistorySummariesMatchRecursion* predicate, and the latter predicate is satisfied, the former predicate is satisfied.

⟨6⟩3. QED

BY ⟨6⟩1, ⟨6⟩2

The third precondition is that in the Memoir-Opt spec, the logical history summary in the *NVRAM* and *SPCR* equals the checkpointed new checkpointed history summary. This follows from a straightforward expansion of definitions, given that an extension is not in progress when an *LL2RepeatOperation* action is executed.

⟨5⟩3. $LL2NVRAMLogicalHistorySummary = ll2CheckpointedNewCheckpointedHistorySummary$

⟨6⟩1. $\neg LL2NVRAM.extensionInProgress$

BY ⟨3⟩2

⟨6⟩2. $LL2NVRAMLogicalHistorySummary = ll2CurrentHistorySummary$

⟨7⟩1. $LL2NVRAMLogicalHistorySummary = [$
$anchor \mapsto LL2NVRAM.historySummaryAnchor,$
$extension \mapsto BaseHashValue]$

BY ⟨6⟩1 DEF *LL2NVRAMLogicalHistorySummary*

⟨7⟩2. $ll2CurrentHistorySummary = [$
$anchor \mapsto LL2NVRAM.historySummaryAnchor,$
$extension \mapsto LL2SPCR]$

BY DEF *ll2CurrentHistorySummary*

⟨7⟩3. $LL2SPCR = BaseHashValue$

BY ⟨3⟩2, ⟨6⟩1

⟨7⟩4. QED

BY ⟨7⟩1, ⟨7⟩2, ⟨7⟩3

⟨6⟩3. $ll2CurrentHistorySummary = ll2CheckpointedNewCheckpointedHistorySummary$

BY ⟨4⟩3

⟨6⟩4. QED

BY ⟨6⟩2, ⟨6⟩3

We use the *HistorySummariesMatchAcrossCheckpointLemma* to prove that the fields are equal. This requires proving some types.

⟨5⟩4. QED

⟨6⟩1. $LL1NVRAM.historySummary \in HashType$

BY ⟨2⟩1 DEF *LL2Refinement*, *LL1TrustedStorageType*

⟨6⟩2. $Hash(LL1RAM.historySummary, input) \in HashType$

BY ⟨4⟩2

⟨6⟩3. $LL2NVRAMLogicalHistorySummary \in HistorySummaryType$

BY ⟨2⟩1, *LL2NVRAMLogicalHistorySummaryTypeSafe*

⟨6⟩4. $ll2NewCheckpointedHistorySummary \in HistorySummaryType$

BY ⟨3⟩3

⟨6⟩5. $LL2NVRAM.hashBarrier \in HashType$

BY ⟨2⟩1, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*

⟨6⟩6. QED

BY ⟨5⟩1, ⟨5⟩2, ⟨5⟩3, ⟨6⟩1, ⟨6⟩2, ⟨6⟩3, ⟨6⟩4, ⟨6⟩5,
HistorySummariesMatchAcrossCheckpointLemma

DEF *ll2CheckpointedNewCheckpointedHistorySummary*

The two cases are exhaustive.

⟨4⟩5. QED

BY ⟨4⟩1, ⟨4⟩3, ⟨4⟩4

The third conjunct in *LL1RepeatOperation*. This conjunct asserts a record equality, so we prove each field in the record separately.

⟨3⟩15. $LL1RAM' = [$
$publicState \mapsto ll1sResult.newPublicState,$
$privateStateEnc \mapsto ll1NewPrivateStateEnc,$
$historySummary \mapsto LL1NVRAM.historySummary,$
$authenticator \mapsto ll1NewAuthenticator]$

The public state field in the primed Memoir-Basic RAM equals the public state in the result of the service, because the primed RAM variables match across the two specs.

⟨4⟩1. $LL1RAM.publicState' = ll1sResult.newPublicState$

⟨5⟩1. $LL1RAM.publicState' = LL2RAM.publicState'$

BY ⟨2⟩1 DEF *LL2Refinement*

⟨5⟩2. $LL2RAM.publicState' = ll2SResult.newPublicState$

BY ⟨3⟩2 DEF *ll2SResult*, *ll2PrivateState*

⟨5⟩3. $ll1sResult.newPublicState = ll2SResult.newPublicState$

BY ⟨3⟩8

⟨5⟩4. QED

BY ⟨5⟩1, ⟨5⟩2, ⟨5⟩3

The encrypted private state field in the primed Memoir-Basic RAM equals the encrypted private state in the result of the service, because the primed RAM variables match across the two specs.

⟨4⟩2. $LL1RAM.privateStateEnc' = ll1NewPrivateStateEnc$

⟨5⟩1. $LL1RAM.privateStateEnc' = LL2RAM.privateStateEnc'$

BY ⟨2⟩1 DEF *LL2Refinement*

⟨5⟩2. $LL2RAM.privateStateEnc' = ll2NewPrivateStateEnc$

BY ⟨3⟩2 DEF *ll2NewPrivateStateEnc*, *ll2SResult*, *ll2PrivateState*

⟨5⟩3. $ll1NewPrivateStateEnc = ll2NewPrivateStateEnc$

BY ⟨3⟩9

⟨5⟩4. QED

BY ⟨5⟩1, ⟨5⟩2, ⟨5⟩3

In the Memoir-Basic spec, the history summary field in the primed RAM equals the history summary in the unprimed $NVRAM$. This is because (1) the history summaries in the RAM variables match across the specs by refinement, (2) the history summary in the primed Memoir-Opt RAM is set equal to the $ll2CurrentHistorySummary$ by the $LL2RepeatOperation$ action, and (3) the history summary in the Memoir-Basic $NVRAM$ matches the $ll2CurrentHistorySummary$ in the Memoir-Opt spec, as we proved above. We can thus use the $HistorySummariesMatchUniqueLemma$ to prove the equality.

$\langle 4\rangle 3.$ $LL1RAM.historySummary' = LL1NVRAM.historySummary$

$\langle 5\rangle 1.$ $HistorySummariesMatch($
$LL1RAM.historySummary', LL2RAM.historySummary', LL2NVRAM.hashBarrier')$

BY $\langle 2\rangle 1$ DEF $LL2Refinement$

$\langle 5\rangle 2.$ $LL2RAM.historySummary' = ll2CurrentHistorySummary$

BY $\langle 3\rangle 2$ DEF $ll2CurrentHistorySummary$

$\langle 5\rangle 3.$ $HistorySummariesMatch($
$LL1NVRAM.historySummary,$
$ll2CurrentHistorySummary,$
$LL2NVRAM.hashBarrier')$

BY $\langle 3\rangle 10$

We use the $HistorySummariesMatchUniqueLemma$ to prove that the fields are equal. This requires proving some types.

$\langle 5\rangle 4.$ QED

$\langle 6\rangle 1.$ $LL1RAM.historySummary' \in HashType$

BY $\langle 2\rangle 1$ DEF $LL2Refinement$, $LL1UntrustedStorageType$

$\langle 6\rangle 2.$ $LL1NVRAM.historySummary \in HashType$

BY $\langle 2\rangle 1$ DEF $LL2Refinement$, $LL1TrustedStorageType$

$\langle 6\rangle 3.$ $LL2RAM.historySummary' \in HistorySummaryType$

BY $\langle 2\rangle 1$, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$

$\langle 6\rangle 4.$ $LL2NVRAM.hashBarrier' \in HashType$

BY $\langle 2\rangle 1$, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$

$\langle 6\rangle 5.$ QED

BY $\langle 5\rangle 1$, $\langle 5\rangle 2$, $\langle 5\rangle 3$, $\langle 6\rangle 1$, $\langle 6\rangle 2$, $\langle 6\rangle 3$, $\langle 6\rangle 4$, $HistorySummariesMatchUniqueLemma$

The authenticator field in the primed Memoir-Basic RAM equals the new authenticator defined in the $LL1RepeatOperation$ action, because the authenticators in the RAM variables match across the specs by refinement. We use the $AuthenticatorsMatchUniqueLemma$ to prove the equality.

$\langle 4\rangle 4.$ $LL1RAM.authenticator' = ll1NewAuthenticator$

$\langle 5\rangle 1.$ PICK $symmetricKey \in SymmetricKeyType$:
$AuthenticatorsMatch($
$LL1RAM.authenticator',$
$LL2RAM.authenticator',$
$symmetricKey,$
$LL2NVRAM.hashBarrier')$

BY $\langle 2\rangle 1$ DEF $LL2Refinement$

$\langle 5\rangle 2.$ $LL2RAM.authenticator' = ll2NewAuthenticator$

BY $\langle 3\rangle 2$
DEF $ll2NewAuthenticator$, $ll2NewHistoryStateBinding$, $ll2CurrentHistorySummaryHash$, $ll2NewStateHash$, $ll2NewPrivateStateEnc$, $ll2SResult$, $ll2PrivateState$, $ll2NewHistorySummary$, $ll2CurrentHistorySummary$

$\langle 5\rangle 3.$ $AuthenticatorsMatch($
$ll1NewAuthenticator,$
$ll2NewAuthenticator,$
$LL2NVRAM.symmetricKey',$
$LL2NVRAM.hashBarrier')$

BY $\langle 3\rangle 12$

We use the *AuthenticatorsMatchUniqueLemma* to prove that the fields are equal. This requires proving some types.

$\langle 5 \rangle 4.$ QED

$\langle 6 \rangle 1.$ $LL1RAM.authenticator' \in MACType$

BY $\langle 2 \rangle 1$ DEF $LL2Refinement$, $LL1UntrustedStorageType$

$\langle 6 \rangle 2.$ $ll1NewAuthenticator \in MACType$

BY $\langle 3 \rangle 5$

$\langle 6 \rangle 3.$ $LL2RAM.authenticator' \in MACType$

BY $\langle 2 \rangle 1$, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$

$\langle 6 \rangle 4.$ $symmetricKey \in SymmetricKeyType$

BY $\langle 5 \rangle 1$

$\langle 6 \rangle 5.$ $LL2NVRAM.symmetricKey' \in SymmetricKeyType$

BY $\langle 2 \rangle 1$, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$

$\langle 6 \rangle 6.$ $LL2NVRAM.hashBarrier' \in HashType$

BY $\langle 2 \rangle 1$, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$

$\langle 6 \rangle 7.$ QED

BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$, $\langle 5 \rangle 3$, $\langle 6 \rangle 1$, $\langle 6 \rangle 2$, $\langle 6 \rangle 3$, $\langle 6 \rangle 4$, $\langle 6 \rangle 5$, $\langle 6 \rangle 6$, $AuthenticatorsMatchUniqueLemma$

The refinement asserts that the RAM record has the appropriate type.

$\langle 4 \rangle 5.$ $LL1RAM' \in LL1UntrustedStorageType$

BY $\langle 2 \rangle 1$ DEF $LL2Refinement$

We use the *LL1RAMRecordCompositionLemma* to unify the field equalities into a record equality.

$\langle 4 \rangle 6.$ QED

BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$, $\langle 4 \rangle 3$, $\langle 4 \rangle 4$, $\langle 4 \rangle 5$, $LL1RAMRecordCompositionLemma$

The fourth conjunct in *LL1RepeatOperation*. The set of observed outputs is equal across the two specs.

$\langle 3 \rangle 16.$ $LL1ObservedOutputs' = LL1ObservedOutputs \cup \{ll1sResult.output\}$

$\langle 4 \rangle 1.$ $LL1ObservedOutputs = LL2ObservedOutputs$

BY $\langle 2 \rangle 1$ DEF $LL2Refinement$

$\langle 4 \rangle 2.$ $LL1ObservedOutputs' = LL2ObservedOutputs'$

BY $\langle 2 \rangle 1$ DEF $LL2Refinement$

$\langle 4 \rangle 3.$ $ll1sResult.output = ll2SResult.output$

BY $\langle 3 \rangle 8$

$\langle 4 \rangle 4.$ $LL2ObservedOutputs' = LL2ObservedOutputs \cup \{ll2SResult.output\}$

BY $\langle 3 \rangle 2$ DEF $ll2SResult$, $ll2PrivateState$

$\langle 4 \rangle 5.$ QED

BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$, $\langle 4 \rangle 3$, $\langle 4 \rangle 4$

The fifth conjunct in *LL1RepeatOperation*. The *NVRAM* is unchanged by the *UnchangedNVRAMLemma*.

$\langle 3 \rangle 17.$ UNCHANGED $LL1NVRAM$

BY $\langle 2 \rangle 1$, $\langle 3 \rangle 2$, $UnchangedNVRAMLemma$

The sixth conjunct in *LL1RepeatOperation*. The disk is unchanged by the *UnchangedDiskLemma*.

$\langle 3 \rangle 18.$ UNCHANGED $LL1Disk$

BY $\langle 2 \rangle 1$, $\langle 3 \rangle 2$, $UnchangedDiskLemma$

The seventh conjunct in *LL1RepeatOperation*. The set of available inputs is unchanged by the *UnchangedAvailableInputsLemma*.

$\langle 3 \rangle 19.$ UNCHANGED $LL1AvailableInputs$

BY $\langle 2 \rangle 1$, $\langle 3 \rangle 2$, $UnchangedAvailableInputsLemma$

The eighth conjunct in *LL1RepeatOperation*. We prove that the primed set of observed authenticators matches across the specs, that the unprimed set of observed authenticators matches across the specs, and that the new authenticator matches across the specs. The *AuthenticatorSetsMatchUniqueLemma* then proves the equality.

$\langle 3 \rangle 20.$ $LL1ObservedAuthenticators' =$

$LL1ObservedAuthenticators \cup \{ll1NewAuthenticator\}$

The primed set of observed authenticators matches across the specs. This follows directly from the refinement.

$\langle 4 \rangle 1.$ $AuthenticatorSetsMatch($
$LL1ObservedAuthenticators'$,
$LL2ObservedAuthenticators'$,
$LL2NVRAM.symmetricKey'$,
$LL2NVRAM.hashBarrier')$
BY $\langle 2 \rangle 1$ DEF $LL2Refinement$

The union matches across the specs. We prove this by proving the matching of each constituent set.

$\langle 4 \rangle 2.$ $AuthenticatorSetsMatch($
$LL1ObservedAuthenticators \cup \{ll1NewAuthenticator\}$,
$LL2ObservedAuthenticators \cup \{ll2NewAuthenticator\}$,
$LL2NVRAM.symmetricKey'$,
$LL2NVRAM.hashBarrier')$
$\langle 5 \rangle 1.$ UNCHANGED $\langle LL2NVRAM.symmetricKey, LL2NVRAM.hashBarrier \rangle$
BY $\langle 3 \rangle 2$

The unprimed set of observed authenticators matches across the specs. This follows directly from the refinement.

$\langle 5 \rangle 2.$ $AuthenticatorSetsMatch($
$LL1ObservedAuthenticators$,
$LL2ObservedAuthenticators$,
$LL2NVRAM.symmetricKey'$,
$LL2NVRAM.hashBarrier')$
BY $\langle 2 \rangle 1$, $\langle 5 \rangle 1$ DEF $LL2Refinement$

The new authenticator matches across the specs. This follows directly from the refinement.

$\langle 5 \rangle 3.$ $AuthenticatorsMatch($
$ll1NewAuthenticator$,
$ll2NewAuthenticator$,
$LL2NVRAM.symmetricKey'$,
$LL2NVRAM.hashBarrier')$
BY $\langle 2 \rangle 1$, $\langle 5 \rangle 1$ DEF $LL2Refinement$
$\langle 5 \rangle 4.$ QED
BY $\langle 5 \rangle 2$, $\langle 5 \rangle 3$ DEF $AuthenticatorSetsMatch$

In the Memoir-Opt spec, the primed set of observed authenticators is formed from the union of the unprimed set of observed authenticators and the new authenticator.

$\langle 4 \rangle 3.$ $LL2ObservedAuthenticators' =$
$LL2ObservedAuthenticators \cup \{ll2NewAuthenticator\}$
BY $\langle 3 \rangle 2$
DEF $ll2NewAuthenticator$, $ll2NewHistoryStateBinding$, $ll2CurrentHistorySummaryHash$, $ll2NewStateHash$, $ll2NewPrivateStateEnc$, $ll2SResult$, $ll2PrivateState$, $ll2NewHistorySummary$, $ll2CurrentHistorySummary$
$\langle 4 \rangle 4.$ QED

We use the *AuthenticatorSetsMatchUniqueLemma* to prove that the sets are equal. This requires proving some types.

$\langle 5 \rangle 1.$ $LL1ObservedAuthenticators' \in$ SUBSET $MACType$
BY $\langle 2 \rangle 1$ DEF $LL2Refinement$, $LL1UntrustedStorageType$
$\langle 5 \rangle 2.$ $LL1ObservedAuthenticators \cup \{ll1NewAuthenticator\} \in$
SUBSET $MACType$
$\langle 6 \rangle 1.$ $LL1ObservedAuthenticators \in$ SUBSET $MACType$
BY $\langle 2 \rangle 1$ DEF $LL2Refinement$, $LL1UntrustedStorageType$
$\langle 6 \rangle 2.$ $ll1NewAuthenticator \in MACType$
BY $\langle 3 \rangle 5$

$\langle 6\rangle 3.$ QED
BY $\langle 6\rangle 1$, $\langle 6\rangle 2$
$\langle 5\rangle 3.$ $LL2ObservedAuthenticators' \in$ SUBSET $MACType$
BY $\langle 2\rangle 1$ DEF $LL2TypeInvariant$
$\langle 5\rangle 4.$ $LL2NVRAM.symmetricKey' \in SymmetricKeyType$
BY $\langle 2\rangle 1$, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$
$\langle 5\rangle 5.$ $LL2NVRAM.hashBarrier' \in HashType$
BY $\langle 2\rangle 1$, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$
$\langle 5\rangle 6.$ QED
BY $\langle 4\rangle 1$, $\langle 4\rangle 2$, $\langle 4\rangle 3$, $\langle 5\rangle 1$, $\langle 5\rangle 2$, $\langle 5\rangle 3$, $\langle 5\rangle 4$, $\langle 5\rangle 5$,
$AuthenticatorSetsMatchUniqueLemma$
$\langle 3\rangle 21.$ QED
BY $\langle 3\rangle 4$, $\langle 3\rangle 13$, $\langle 3\rangle 14$, $\langle 3\rangle 15$, $\langle 3\rangle 16$, $\langle 3\rangle 17$, $\langle 3\rangle 18$, $\langle 3\rangle 19$, $\langle 3\rangle 20$
DEF $LL1RepeatOperation$, $ll1StateHash$, $ll1HistoryStateBinding$, $ll1PrivateState$,
$ll1sResult$, $ll1NewPrivateStateEnc$, $ll1NewStateHash$,
$ll1NewHistoryStateBinding$, $ll1NewAuthenticator$

A Memoir-Opt $LL2TakeCheckpoint$ action refines to a Memoir-Basic stuttering step. This is not at all obvious, and it is the essence of why the enhancement from Memoir-Basic to Memoir-Opt spec continues to satisfy the implementation.

$\langle 2\rangle 6.$ $LL2TakeCheckpoint \Rightarrow$ UNCHANGED $LL1Vars$
$\langle 3\rangle 1.$ HAVE $LL2TakeCheckpoint$

We re-state the definition from $LL2TakeCheckpoint$.

$\langle 3\rangle$ $newHistorySummaryAnchor \triangleq Hash(LL2NVRAM.historySummaryAnchor, LL2SPCR)$

We then hide the definition.

$\langle 3\rangle$ HIDE DEF $newHistorySummaryAnchor$

Two frequently useful facts are that the symmetric key and the hash barrier in the Memoir-Opt $NVRAM$ are unchanged. This follows directly from the definition of $LL2TakeCheckpoint$.

$\langle 3\rangle 2.$ $\wedge$ UNCHANGED $LL2NVRAM.symmetricKey$
$\wedge$ UNCHANGED $LL2NVRAM.hashBarrier$
$\langle 4\rangle 1.$ $LL2NVRAM' = [$
$historySummaryAnchor \mapsto newHistorySummaryAnchor$,
$symmetricKey \mapsto LL2NVRAM.symmetricKey$,
$hashBarrier \mapsto LL2NVRAM.hashBarrier$,
$extensionInProgress \mapsto$ FALSE$]$
BY $\langle 3\rangle 1$ DEF $LL2TakeCheckpoint$, $newHistorySummaryAnchor$
$\langle 4\rangle 2.$ QED
BY $\langle 4\rangle 1$ DEF $LL2TrustedStorageType$

We prove the UNCHANGED status for each Memoir-Basic variable in turn. For $LL1AvailableInputs$, $LL1ObservedOutputs$, $LL1ObservedAuthenticators$, $LL1Disk$, and $LL1RAM$, the UNCHANGED status follows directly from the lemmas we have proven for this purpose.

$\langle 3\rangle 3.$ UNCHANGED $LL1AvailableInputs$
BY $\langle 2\rangle 1$, $\langle 3\rangle 1$, $\langle 3\rangle 2$, $UnchangedAvailableInputsLemma$DEF $LL2TakeCheckpoint$
$\langle 3\rangle 4.$ UNCHANGED $LL1ObservedOutputs$
BY $\langle 2\rangle 1$, $\langle 3\rangle 1$, $\langle 3\rangle 2$, $UnchangedObservedOutputsLemma$DEF $LL2TakeCheckpoint$
$\langle 3\rangle 5.$ UNCHANGED $LL1ObservedAuthenticators$
BY $\langle 2\rangle 1$, $\langle 3\rangle 1$, $\langle 3\rangle 2$, $UnchangedObservedAuthenticatorsLemma$DEF $LL2TakeCheckpoint$
$\langle 3\rangle 6.$ UNCHANGED $LL1Disk$
BY $\langle 2\rangle 1$, $\langle 3\rangle 1$, $\langle 3\rangle 2$, $UnchangedDiskLemma$DEF $LL2TakeCheckpoint$
$\langle 3\rangle 7.$ UNCHANGED $LL1RAM$
BY $\langle 2\rangle 1$, $\langle 3\rangle 1$, $\langle 3\rangle 2$, $UnchangedRAMLemma$DEF $LL2TakeCheckpoint$

To prove the UNCHANGED status of the Memoir-Basic $NVRAM$ value, we prove each of the two fields separately.

$\langle 3\rangle 8.$ UNCHANGED $LL1NVRAM$

Proving the UNCHANGED status of the history summary in the Memoir-Basic *NVRAM* is quite involved. The top level is straightforward. We simply prove that both the unprimed and primed history summaries in the Memoir-Basic *NVRAM* match the primed logical history summary in the Memoir-Opt *NVRAM* and *SPCR*. Then, from the *HistorySummariesMatchUniqueLemma*, we conclude that both unprimed and unprimed history summaries in the Memoir-Basic *NVRAM* are equal.

$\langle 4 \rangle 2.$ UNCHANGED $LL1NVRAM.historySummary$

The primed history summary in the Memoir-Basic *NVRAM* matches the primed logical history summary in the Memoir-Opt *NVRAM* and *SPCR*. This follows directly from the refinement, given that the hash barrier in the Memoir-Opt *NVRAM* has not changed.

$\langle 5 \rangle 1.$ $HistorySummariesMatch($
$\quad LL1NVRAM.historySummary',$
$\quad LL2NVRAMLogicalHistorySummary',$
$\quad LL2NVRAM.hashBarrier)$
BY $\langle 3 \rangle 2$, $\langle 2 \rangle 1$ DEF $LL2Refinement$

The unprimed history summary in the Memoir-Basic *NVRAM* matches the primed logical history summary in the Memoir-Opt *NVRAM* and *SPCR*. The proof involves three main steps.

$\langle 5 \rangle 2.$ $HistorySummariesMatch($
$\quad LL1NVRAM.historySummary,$
$\quad LL2NVRAMLogicalHistorySummary',$
$\quad LL2NVRAM.hashBarrier)$

$\langle 6 \rangle$ $ll2InitialHistorySummary \triangleq [anchor \mapsto BaseHashValue, extension \mapsto BaseHashValue]$

First, we pick a set of variables that satisfy the existentials in the *HistorySummariesMatchRecursion* operator, for the unprimed state. This is more involved than might seem necessary.

$\langle 6 \rangle 1.$ PICK $input \in InputType,$
$\quad previousLL1HistorySummary \in HashType,$
$\quad previousLL2HistorySummary \in HistorySummaryType :$
$\quad HistorySummariesMatchRecursion($
$\quad\quad LL1NVRAM.historySummary,$
$\quad\quad LL2NVRAMLogicalHistorySummary,$
$\quad\quad LL2NVRAM.hashBarrier)!($
$\quad\quad input,$
$\quad\quad previousLL1HistorySummary,$
$\quad\quad previousLL2HistorySummary)$

First, we assert that the unprimed history summaries match across the two specs, which follows from the refinement.

$\langle 7 \rangle 1.$ $HistorySummariesMatch($
$\quad LL1NVRAM.historySummary,$
$\quad LL2NVRAMLogicalHistorySummary,$
$\quad LL2NVRAM.hashBarrier)$
BY $\langle 2 \rangle 1$ DEF $LL2Refinement$

Then, we prove some types, to satisfy the universal quantifiers in *HistorySummariesMatchDefinition*.

$\langle 7 \rangle 2.$ $LL1NVRAM.historySummary \in HashType$
BY $\langle 2 \rangle 1$ DEF $LL2Refinement$, $LL1TrustedStorageType$

$\langle 7 \rangle 3.$ $LL2NVRAMLogicalHistorySummary \in HistorySummaryType$
BY $\langle 2 \rangle 1$, $LL2NVRAMLogicalHistorySummaryTypeSafe$

$\langle 7 \rangle 4.$ $LL2NVRAM.hashBarrier \in HashType$
BY $\langle 2 \rangle 1$, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$

We prove that the Memoir-Opt logical history summary does not equal the initial history summary. This follows from an enablement condition in the *LL2TakeCheckpoint* action, namely that *LL2SPCR* does not equal *BaseHashValue*.

$\langle 7 \rangle 5.$ $LL2NVRAMLogicalHistorySummary \neq ll2InitialHistorySummary$

$\langle 8 \rangle 1.$ $LL2NVRAMLogicalHistorySummary.extension = LL2SPCR$

⟨9⟩1. $LL2NVRAMLogicalHistorySummary = [$
$anchor \mapsto LL2NVRAM.historySummaryAnchor,$
$extension \mapsto LL2SPCR]$
⟨10⟩1. $LL2NVRAM.extensionInProgress = \text{TRUE}$
BY ⟨3⟩1 DEF *LL2TakeCheckpoint*
⟨10⟩2. $LL2SPCR \neq BaseHashValue$
BY ⟨3⟩1, ⟨10⟩1 DEF *LL2TakeCheckpoint*
⟨10⟩3. QED
BY ⟨10⟩1, ⟨10⟩2 DEF *LL2NVRAMLogicalHistorySummary*
⟨9⟩2. QED
BY ⟨9⟩1
⟨8⟩2. $LL2SPCR \neq BaseHashValue$
BY ⟨3⟩1 DEF *LL2TakeCheckpoint*
⟨8⟩3. $ll2InitialHistorySummary.extension = BaseHashValue$
BY DEF *ll2InitialHistorySummary*
⟨8⟩4. QED
BY ⟨8⟩1, ⟨8⟩2, ⟨8⟩3

Finally from *HistorySummariesMatchDefinition*, we can conclude that the quantified *HistorySummariesMatchRecursion* predicate is satisfied.

⟨7⟩6. QED
BY ⟨7⟩1, ⟨7⟩2, ⟨7⟩3, ⟨7⟩4, ⟨7⟩5, *HistorySummariesMatchDefinition*
DEF *HistorySummariesMatchRecursion*

Second, we prove that the quantified *HistorySummariesMatchRecursion* predicate is satisfied for the unprimed history summary in the Memoir-Basic spec and the primed logical history summary in the Memoir-Opt spec.

⟨6⟩2. $HistorySummariesMatchRecursion($
$LL1NVRAM.historySummary,$
$LL2NVRAMLogicalHistorySummary',$
$LL2NVRAM.hashBarrier)!($
$input,$
$previousLL1HistorySummary,$
$previousLL2HistorySummary)$

The quantified predicate is satisfied for the unprimed variables, because we picked appropriate values for the quantifiers above.

⟨7⟩1. $HistorySummariesMatchRecursion($
$LL1NVRAM.historySummary,$
$LL2NVRAMLogicalHistorySummary,$
$LL2NVRAM.hashBarrier)!($
$input,$
$previousLL1HistorySummary,$
$previousLL2HistorySummary)$
BY ⟨6⟩1

We expand the definition of the quantified predicate into its three constituent conjuncts. The first two conjuncts follow directly from the quantified *HistorySummariesMatchRecursion* predicate for the unprimed history summaries.

⟨7⟩2. $HistorySummariesMatch($
$previousLL1HistorySummary,$
$previousLL2HistorySummary,$
$LL2NVRAM.hashBarrier)$
BY ⟨7⟩1
⟨7⟩3. $LL1NVRAM.historySummary = Hash(previousLL1HistorySummary, input)$
BY ⟨7⟩1

We expand the definition of the *LL2HistorySummaryIsSuccessor* predicate and ignore the first disjunct, since we know that the history summary has been checkpointed.

⟨7⟩4. *LL2HistorySummaryIsSuccessor*(
LL2NVRAMLogicalHistorySummary′,
previousLL2HistorySummary,
input,
LL2NVRAM.hashBarrier)

⟨8⟩ *successorHistorySummary* $\triangleq$
Successor(*previousLL2HistorySummary*, *input*, *LL2NVRAM.hashBarrier*)

⟨8⟩ *checkpointedSuccessorHistorySummary* $\triangleq$ *Checkpoint*(*successorHistorySummary*)

⟨8⟩ HIDE DEF *successorHistorySummary*, *checkpointedSuccessorHistorySummary*

All of the work is in proving the second disjunct of the *LL2HistorySummaryIsSuccessor* predicate.

⟨8⟩1. *LL2NVRAMLogicalHistorySummary*′ = *checkpointedSuccessorHistorySummary*

First, we prove that the *LL2NVRAMLogicalHistorySummary* equals a record with the fields of new history summary anchor and base hash value. This follows fairly directly from the definitions of *LL2NVRAMLogicalHistorySummary* and *LL2TakeCheckpoint*.

⟨9⟩1. *LL2NVRAMLogicalHistorySummary*′ = [
anchor ↦ *newHistorySummaryAnchor*,
extension ↦ *BaseHashValue*]

⟨10⟩1. *LL2NVRAM*′ = [
historySummaryAnchor ↦ *newHistorySummaryAnchor*,
symmetricKey ↦ *LL2NVRAM.symmetricKey*,
hashBarrier ↦ *LL2NVRAM.hashBarrier*,
extensionInProgress ↦ FALSE]

BY ⟨3⟩1 DEF *LL2TakeCheckpoint*, *newHistorySummaryAnchor*

⟨10⟩2. *LL2NVRAMLogicalHistorySummary*′ = [
anchor ↦ *LL2NVRAM.historySummaryAnchor*′,
extension ↦ *BaseHashValue*]

⟨11⟩1. *LL2NVRAM.extensionInProgress*′ = FALSE

BY ⟨10⟩1

⟨11⟩2. QED

BY ⟨11⟩1 DEF *LL2NVRAMLogicalHistorySummary*

⟨10⟩3. *LL2NVRAM.historySummaryAnchor*′ = *newHistorySummaryAnchor*

BY ⟨10⟩1

⟨10⟩4. QED

BY ⟨10⟩2, ⟨10⟩3

Second, we prove that the *checkpointedSuccessorHistorySummary* also equals a record with the fields of new history summary anchor and base hash value.

⟨9⟩2. *checkpointedSuccessorHistorySummary* = [
anchor ↦ *newHistorySummaryAnchor*,
extension ↦ *BaseHashValue*]

The main step is proving that the successor history summary equals the logical history summary in the Memoir-Opt *NVRAM* and *SPCR*.

⟨10⟩1. *successorHistorySummary* = *LL2NVRAMLogicalHistorySummary*

From the *LL2HistorySummaryIsSuccessor* predicate, we know that the logical history summary must either equal the successor or the checkpointed successor.

⟨11⟩1. ∨ *LL2NVRAMLogicalHistorySummary* = *successorHistorySummary*
∨ *LL2NVRAMLogicalHistorySummary* = *checkpointedSuccessorHistorySummary*

⟨12⟩1. *LL2HistorySummaryIsSuccessor*(
LL2NVRAMLogicalHistorySummary,
previousLL2HistorySummary,

$input$,
$LL2NVRAM.hashBarrier$)

BY ⟨7⟩1

⟨12⟩2. QED

BY ⟨12⟩1

DEF $LL2HistorySummaryIsSuccessor$, $successorHistorySummary$, $checkpointedSuccessorHistorySummary$

The logical history summary does not equal the checkpointed successor.

⟨11⟩2. $LL2NVRAMLogicalHistorySummary \neq checkpointedSuccessorHistorySummary$

The extension field of the logical history summary does not equal the base hash value. This follows from an enablement condition of the *TakeCheckpoint* action.

⟨12⟩1. $LL2NVRAMLogicalHistorySummary.extension \neq BaseHashValue$

⟨13⟩1. $LL2NVRAMLogicalHistorySummary.extension = LL2SPCR$

⟨14⟩1. $LL2NVRAMLogicalHistorySummary = [$
$anchor \mapsto LL2NVRAM.historySummaryAnchor,$
$extension \mapsto LL2SPCR]$

⟨15⟩1. $LL2NVRAM.extensionInProgress = \text{TRUE}$

BY ⟨3⟩1 DEF $LL2TakeCheckpoint$

⟨15⟩2. $LL2SPCR \neq BaseHashValue$

BY ⟨3⟩1, ⟨15⟩1 DEF $LL2TakeCheckpoint$

⟨15⟩3. QED

BY ⟨15⟩1, ⟨15⟩2 DEF $LL2NVRAMLogicalHistorySummary$

⟨14⟩2. QED

BY ⟨14⟩1

⟨13⟩2. $LL2SPCR \neq BaseHashValue$

BY ⟨3⟩1 DEF $LL2TakeCheckpoint$

⟨13⟩3. QED

BY ⟨13⟩1, ⟨13⟩2

The extension field of the checkpointed successor does equal the base hash value. This follows from the *CheckpointHasBaseExtensionLemma*.

⟨12⟩2. $checkpointedSuccessorHistorySummary.extension = BaseHashValue$

⟨13⟩1. $successorHistorySummary \in HistorySummaryType$

⟨14⟩1. $previousLL2HistorySummary \in HistorySummaryType$

BY ⟨6⟩1

⟨14⟩2. $input \in InputType$

BY ⟨6⟩1

⟨14⟩3. $LL2NVRAM.hashBarrier \in HashType$

BY ⟨2⟩1, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$

⟨14⟩4. QED

BY ⟨14⟩1, ⟨14⟩2, ⟨14⟩3, $SuccessorTypeSafe$DEF $successorHistorySummary$

⟨13⟩2. QED

BY ⟨13⟩1, $CheckpointHasBaseExtensionLemma$

DEF $checkpointedSuccessorHistorySummary$

Since he extension field of the logical history summary does not equal the base hash value, but the extension field of the checkpointed successor does equal the base hash value, it follows that these two records are unequal.

⟨12⟩3. QED

BY ⟨12⟩1, ⟨12⟩2

⟨11⟩3. QED

BY ⟨11⟩1, ⟨11⟩2

Given that the successor history summary equals the logical history summary in the Memoir-Opt *NVRAM* and *SPCR*, we can readily derive the specific field values for this record.

$\langle 10\rangle 2.$ $successorHistorySummary = [$
$anchor \mapsto LL2NVRAM.historySummaryAnchor,$
$extension \mapsto LL2SPCR]$

$\langle 11\rangle 1.$ $LL2NVRAMLogicalHistorySummary = [$
$anchor \mapsto LL2NVRAM.historySummaryAnchor,$
$extension \mapsto LL2SPCR]$

$\langle 12\rangle 1.$ $LL2NVRAM.extensionInProgress = \text{TRUE}$
BY $\langle 3\rangle 1$ DEF $LL2TakeCheckpoint$

$\langle 12\rangle 2.$ $LL2SPCR \neq BaseHashValue$
BY $\langle 3\rangle 1$, $\langle 12\rangle 1$ DEF $LL2TakeCheckpoint$

$\langle 12\rangle 3.$ QED
BY $\langle 12\rangle 1$, $\langle 12\rangle 2$ DEF $LL2NVRAMLogicalHistorySummary$

$\langle 11\rangle 2.$ QED
BY $\langle 10\rangle 1$, $\langle 11\rangle 1$

Now, we merely need to show that the the extension field is not equal to the base hash value, so the definition of the *Checkpoint* operator yields the same value as the *TakeCheckpoint* action.

$\langle 10\rangle 3.$ QED

$\langle 11\rangle 1.$ $successorHistorySummary.extension \neq BaseHashValue$

$\langle 12\rangle 1.$ $successorHistorySummary.extension = LL2SPCR$
BY $\langle 10\rangle 2$

$\langle 12\rangle 2.$ $LL2SPCR \neq BaseHashValue$
BY $\langle 3\rangle 1$ DEF $LL2TakeCheckpoint$

$\langle 12\rangle 3.$ QED
BY $\langle 12\rangle 1$, $\langle 12\rangle 2$

$\langle 11\rangle 2.$ QED
BY $\langle 10\rangle 2$, $\langle 11\rangle 1$
DEF $checkpointedSuccessorHistorySummary$, $Checkpoint$, $newHistorySummaryAnchor$

Since the *LL2NVRAMLogicalHistorySummary* and the *checkpointedSuccessorHistorySummary* each equal the same value, they equal each other.

$\langle 9\rangle 3.$ QED
BY $\langle 9\rangle 1$, $\langle 9\rangle 2$

$\langle 8\rangle 2.$ QED
BY $\langle 8\rangle 1$
DEF $LL2HistorySummaryIsSuccessor$, $checkpointedSuccessorHistorySummary$, $successorHistorySummary$

The predicate is satisfied because each of its conjuncts is satisfied.

$\langle 7\rangle 5.$ QED
BY $\langle 7\rangle 2$, $\langle 7\rangle 3$, $\langle 7\rangle 4$

Third, we prove that for the unprimed Memoir-Basic history summary and the primed Memoir-Opt logical history summary, the *HistorySummariesMatch* predicate equals the quantified *HistorySummariesMatchRecursion* predicate.

$\langle 6\rangle 3.$ $HistorySummariesMatch($
$LL1NVRAM.historySummary,$
$LL2NVRAMLogicalHistorySummary',$
$LL2NVRAM.hashBarrier)$
$= HistorySummariesMatchRecursion($
$LL1NVRAM.historySummary,$
$LL2NVRAMLogicalHistorySummary',$
$LL2NVRAM.hashBarrier)$

We prove some types, to satisfy the universal quantifiers in *HistorySummariesMatchDefinition*.

$\langle 7\rangle 1.$ $LL1NVRAM.historySummary \in HashType$

BY $\langle 2\rangle 1$ DEF *LL2Refinement*, *LL1TrustedStorageType*

$\langle 7\rangle 2.$ $LL2NVRAMLogicalHistorySummary' \in HistorySummaryType$

BY $\langle 2\rangle 1$, *LL2NVRAMLogicalHistorySummaryTypeSafe*

$\langle 7\rangle 3.$ $LL2NVRAM.hashBarrier \in HashType$

BY $\langle 2\rangle 1$, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*

We prove that the Memoir-Opt logical history summary does not equal the initial history summary. There are three sub-steps.

$\langle 7\rangle 4.$ $LL2NVRAMLogicalHistorySummary' \neq ll2InitialHistorySummary$

First, we prove that the primed value of the anchor field in the Memoir-Opt logical history summary equals the new history summary anchor defined in the *TakeCheckpoint* action. This follows fairly directly from the definitions of the *TakeCheckpoint* action and the *LL2NVRAMLogicalHistorySummary* operator.

$\langle 8\rangle 1.$ $LL2NVRAMLogicalHistorySummary.anchor' = newHistorySummaryAnchor$

$\langle 9\rangle 1.$ $LL2NVRAM' = [$
$historySummaryAnchor \mapsto newHistorySummaryAnchor,$
$symmetricKey \mapsto LL2NVRAM.symmetricKey,$
$hashBarrier \mapsto LL2NVRAM.hashBarrier,$
$extensionInProgress \mapsto \text{FALSE}]$

BY $\langle 3\rangle 1$ DEF *LL2TakeCheckpoint*, *newHistorySummaryAnchor*

$\langle 9\rangle 2.$ $LL2NVRAMLogicalHistorySummary' = [$
$anchor \mapsto LL2NVRAM.historySummaryAnchor',$
$extension \mapsto BaseHashValue]$

$\langle 10\rangle 1.$ $LL2NVRAM.extensionInProgress' = \text{FALSE}$

BY $\langle 9\rangle 1$

$\langle 10\rangle 2.$ QED

BY $\langle 10\rangle 1$ DEF *LL2NVRAMLogicalHistorySummary*

$\langle 9\rangle 3.$ $LL2NVRAM.historySummaryAnchor' = newHistorySummaryAnchor$

BY $\langle 9\rangle 1$

$\langle 9\rangle 4.$ QED

BY $\langle 9\rangle 2$, $\langle 9\rangle 3$

Second, we prove that the new history summary anchor is not equal to the base hash value. This follows because the new history summary anchor is generated as a hash by the *TakeCheckpoint* action, and the *BaseHashValueUnique* property tells us that no hash value generated by the *Hash* function can equal the base hash value.

$\langle 8\rangle 2.$ $newHistorySummaryAnchor \neq BaseHashValue$

$\langle 9\rangle 1.$ $LL2NVRAM.historySummaryAnchor \in HashDomain$

$\langle 10\rangle 1.$ $LL2NVRAM.historySummaryAnchor \in HashType$

BY $\langle 2\rangle 1$, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*

$\langle 10\rangle 2.$ QED

BY $\langle 10\rangle 1$ DEF *HashDomain*

$\langle 9\rangle 2.$ $LL2SPCR \in HashDomain$

$\langle 10\rangle 1.$ $LL2SPCR \in HashType$

BY $\langle 2\rangle 1$ DEF *LL2TypeInvariant*

$\langle 10\rangle 2.$ QED

BY $\langle 10\rangle 1$ DEF *HashDomain*

$\langle 9\rangle 3.$ QED

BY $\langle 9\rangle 1$, $\langle 9\rangle 2$, *BaseHashValueUnique*DEF *newHistorySummaryAnchor*

Third, the anchor field of the initial history summary equals the base hash value, by definition.

$\langle 8\rangle 3.$ $ll2InitialHistorySummary.anchor = BaseHashValue$

BY DEF *ll2InitialHistorySummary*

$\langle 8\rangle 4.$ QED

BY $\langle 8\rangle 1$, $\langle 8\rangle 2$, $\langle 8\rangle 3$

Finally, from *HistorySummariesMatchDefinition*, we can conclude that the *HistorySummariesMatch* predicate equals the quantified *HistorySummariesMatchRecursion* predicate.

$\langle 7\rangle 5$. QED
BY $\langle 7\rangle 1$, $\langle 7\rangle 2$, $\langle 7\rangle 3$, $\langle 7\rangle 4$, *HistorySummariesMatchDefinition*
DEF *ll2InitialHistorySummary*

We tie the above steps together by first deriving the unquantified *HistorySummariesMatchRecursion* predicate from the quantified predicate, and then proving the straightforward equality.

$\langle 6\rangle 4$. QED
$\langle 7\rangle 1$. *HistorySummariesMatchRecursion*(
LL1NVRAM.historySummary,
LL2NVRAMLogicalHistorySummary′,
LL2NVRAM.hashBarrier)
$\langle 8\rangle 1$. $\wedge$ *input* $\in$ *InputType*
$\wedge$ *previousLL1HistorySummary* $\in$ *HashType*
$\wedge$ *previousLL2HistorySummary* $\in$ *HistorySummaryType*
BY $\langle 6\rangle 1$
$\langle 8\rangle 2$. QED
BY $\langle 6\rangle 2$, $\langle 8\rangle 1$ DEF *HistorySummariesMatchRecursion*
$\langle 7\rangle 2$. QED
BY $\langle 6\rangle 2$, $\langle 6\rangle 3$, $\langle 7\rangle 1$

We use the *HistorySummariesMatchUniqueLemma* to conclude that both unprimed and unprimed history summaries in the Memoir-Basic *NVRAM* are equal.

$\langle 5\rangle 3$. QED
$\langle 6\rangle 1$. *LL1NVRAM.historySummary*′ $\in$ *HashType*
BY $\langle 2\rangle 1$ DEF *LL2Refinement*, *LL1TrustedStorageType*
$\langle 6\rangle 2$. *LL1NVRAM.historySummary* $\in$ *HashType*
BY $\langle 2\rangle 1$ DEF *LL2Refinement*, *LL1TrustedStorageType*
$\langle 6\rangle 3$. *LL2NVRAMLogicalHistorySummary*′ $\in$ *HistorySummaryType*
BY $\langle 2\rangle 1$, *LL2NVRAMLogicalHistorySummaryTypeSafe*
$\langle 6\rangle 4$. *LL2NVRAM.hashBarrier* $\in$ *HashType*
BY $\langle 2\rangle 1$, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*
$\langle 6\rangle 5$. QED
BY $\langle 5\rangle 1$, $\langle 5\rangle 2$, $\langle 6\rangle 1$, $\langle 6\rangle 2$, $\langle 6\rangle 3$, $\langle 6\rangle 4$, *HistorySummariesMatchUniqueLemma*

Proving the UNCHANGED status of the symmetric key in the Memoir-Basic *NVRAM* is straightforward, using the lemma we have proven for this purpose.

$\langle 4\rangle 3$. UNCHANGED *LL1NVRAM.symmetricKey*
$\langle 5\rangle 1$. UNCHANGED *LL2NVRAM.symmetricKey*
BY $\langle 3\rangle 2$
$\langle 5\rangle 2$. QED
BY $\langle 2\rangle 1$, $\langle 5\rangle 1$, *UnchangedNVRAMSymmetricKeyLemma*

We can then use the *LL1NVRAMRecordCompositionLemma* directly, once we prove some types.

$\langle 4\rangle 8$. QED
$\langle 5\rangle 1$. *LL1NVRAM* $\in$ *LL1TrustedStorageType*
BY $\langle 2\rangle 1$ DEF *LL2Refinement*
$\langle 5\rangle 2$. *LL1NVRAM*′ $\in$ *LL1TrustedStorageType*
BY $\langle 2\rangle 1$ DEF *LL2Refinement*
$\langle 5\rangle 3$. QED
BY $\langle 4\rangle 2$, $\langle 4\rangle 3$, $\langle 5\rangle 1$, $\langle 5\rangle 2$, *LL1NVRAMRecordCompositionLemma*

All of the Memoir-Basic variables are unchanged.

$\langle 3\rangle 9$. QED
BY $\langle 3\rangle 3$, $\langle 3\rangle 4$, $\langle 3\rangle 5$, $\langle 3\rangle 6$, $\langle 3\rangle 7$, $\langle 3\rangle 8$ DEF *LL1Vars*

A Memoir-Opt *LL2Restart* action refines to one of two actions in the Memoir-Basic spec. If an extension is in progress at the time the *LL2Restart* occurs, the loss of the *SPCR* value caused by the *LL2Restart* is fatal, so this refines to a *LL1RestrictedCorruption* action in the Memoir-Basic spec, which in turn refines to an *HLDie* action in the high-level spec. On the other hand, if an extension is not in progress at the time the *LL2Restart* occurs, the action refines to an *LL1Restart* action in the Memoir-Basic spec.

⟨2⟩7. *LL2Restart* ⇒
IF *LL2NVRAM.extensionInProgress*
THEN
LL1RestrictedCorruption
ELSE
LL1Restart

We assume the antecedent.

⟨3⟩1. HAVE *LL2Restart*

We pick a set of variables of the appropriate types that satisfy the *LL2CorruptRAM* action.

⟨3⟩2. PICK $ll2UntrustedStorage \in LL2UntrustedStorageType$,
$ll2RandomSymmetricKey \in SymmetricKeyType \setminus \{LL2NVRAM.symmetricKey\}$,
$ll2Hash \in HashType$:
$LL2Restart!(ll2UntrustedStorage, ll2RandomSymmetricKey, ll2Hash)$
BY ⟨3⟩1 DEF *LL2Restart*

We first prove that the primed state of the Memoir-Basic RAM has a value that satisfies a particular constraint that is imposed by both the *LL1RestrictedCorruption* action and by the *LL1Restart* action.

⟨3⟩3. $\exists\ ll1UntrustedStorage \in LL1UntrustedStorageType$,
$ll1RandomSymmetricKey \in SymmetricKeyType \setminus \{LL1NVRAM.symmetricKey\}$,
$ll1Hash \in HashType$:
$\land\ ll1UntrustedStorage.authenticator =$
$GenerateMAC(ll1RandomSymmetricKey, ll1Hash)$
$\land\ LL1RAM' = ll1UntrustedStorage$

We pick a symmetric key that satisfies the *AuthenticatorsMatch* predicate for the primed states of the authenticators in the RAM variables of the two specs.

⟨4⟩3. PICK $symmetricKey \in SymmetricKeyType$:
$AuthenticatorsMatch($
$LL1RAM.authenticator'$,
$LL2RAM.authenticator'$,
$symmetricKey$,
$LL2NVRAM.hashBarrier')$
BY ⟨2⟩1 DEF *LL2Refinement*

We pick a set of variables of the appropriate types that satisfy the quantified *AuthenticatorsMatch* predicate.

⟨4⟩4. PICK $stateHash \in HashType$,
$ll1HistorySummary \in HashType$,
$ll2HistorySummary \in HistorySummaryType$:
$AuthenticatorsMatch($
$LL1RAM.authenticator'$,
$LL2RAM.authenticator'$,
$symmetricKey$,
$LL2NVRAM.hashBarrier')!($
$stateHash, ll1HistorySummary, ll2HistorySummary)!1$
BY ⟨4⟩3 DEF *AuthenticatorsMatch*

We re-state the definitions from the LET in *AuthenticatorsMatch*.

⟨4⟩ $ll1HistoryStateBinding \triangleq Hash(ll1HistorySummary, stateHash)$
⟨4⟩ $ll2HistorySummaryHash \triangleq Hash(ll2HistorySummary.anchor, ll2HistorySummary.extension)$
⟨4⟩ $ll2HistoryStateBinding \triangleq Hash(ll2HistorySummaryHash, stateHash)$

We prove the types of the definitions, with help from the *AuthenticatorsMatchDefsTypeSafeLemma*.

$\langle 4\rangle 5.\ \land\ ll1HistoryStateBinding \in HashType$
$\quad\land\ ll2HistorySummaryHash \in HashType$
$\quad\land\ ll2HistoryStateBinding \in HashType$
BY $\langle 4\rangle 4$, *AuthenticatorsMatchDefsTypeSafeLemma*

We hide the definitions.

$\langle 4\rangle$ HIDE DEF *ll1HistoryStateBinding*, *ll2HistorySummaryHash*, *ll2HistoryStateBinding*

To prove the constraints regarding the primed state of the Memoir-Basic RAM, we first prove the types of the three witnesses for the existentially quantified variables.

$\langle 4\rangle 6.\ LL1RAM' \in LL1UntrustedStorageType$
BY $\langle 2\rangle 1$ DEF *LL2Refinement*
$\langle 4\rangle 7.\ ll2RandomSymmetricKey \in SymmetricKeyType \setminus \{LL1NVRAM.symmetricKey\}$
$\langle 5\rangle 1.\ ll2RandomSymmetricKey \in SymmetricKeyType \setminus \{LL2NVRAM.symmetricKey\}$
BY $\langle 3\rangle 2$
$\langle 5\rangle 2.\ LL1NVRAM.symmetricKey = LL2NVRAM.symmetricKey$
BY $\langle 2\rangle 1$ DEF *LL2Refinement*
$\langle 5\rangle 3.$ QED
BY $\langle 5\rangle 1$, $\langle 5\rangle 2$
$\langle 4\rangle 8.\ ll1HistoryStateBinding \in HashType$
BY $\langle 4\rangle 5$

We prove that the authenticator in the Memoir-Basic spec is generated as a *MAC* with a random symmetric key. This follows directly from the *AuthenticatorGeneratedLemma*, once we prove the preconditions for the lemma.

$\langle 4\rangle 9.\ LL1RAM.authenticator' =$
$\quad GenerateMAC(ll2RandomSymmetricKey, ll1HistoryStateBinding)$

We need to prove some types.

$\langle 5\rangle 1.\ stateHash \in HashType$
BY $\langle 4\rangle 4$
$\langle 5\rangle 2.\ ll1HistorySummary \in HashType$
BY $\langle 4\rangle 4$
$\langle 5\rangle 3.\ ll2HistorySummary \in HistorySummaryType$
BY $\langle 4\rangle 4$
$\langle 5\rangle 4.\ LL1RAM.authenticator' \in MACType$
$\langle 6\rangle 1.\ LL1RAM' \in LL1UntrustedStorageType$
BY $\langle 2\rangle 1$ DEF *LL2Refinement*
$\langle 6\rangle 2.$ QED
BY $\langle 6\rangle 1$ DEF *LL1UntrustedStorageType*
$\langle 5\rangle 5.\ LL2RAM.authenticator' \in MACType$
BY $\langle 2\rangle 1$, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*
$\langle 5\rangle 6.\ ll2RandomSymmetricKey \in SymmetricKeyType$
$\langle 6\rangle 1.\ ll2RandomSymmetricKey \in SymmetricKeyType \setminus \{LL2NVRAM.symmetricKey\}$
BY $\langle 3\rangle 2$
$\langle 6\rangle 2.$ QED
BY $\langle 6\rangle 1$
$\langle 5\rangle 7.\ LL2NVRAM.hashBarrier' \in HashType$
BY $\langle 2\rangle 1$, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*

Then we prove the three conjuncts in the antecedent of the *AuthenticatorGeneratedLemma*. The first conjunct follows from a conjunct in the refinement.

$\langle 5\rangle 8.\ HistorySummariesMatch(ll1HistorySummary, ll2HistorySummary, LL2NVRAM.hashBarrier')$
BY $\langle 4\rangle 4$

The second conjunct in the antecedent of the *AuthenticatorGeneratedLemma* mainly follows from the refinement, but we also have to prove that the symmetric key specified by an existential in the refinement matches the random symmetric key specified in the *LL2CorruptRAM* action. This follows from the *MACUnforgeable* property.

$\langle 5 \rangle 9.$ $AuthenticatorsMatch($
$\quad LL1RAM.authenticator',$
$\quad LL2RAM.authenticator',$
$\quad ll2RandomSymmetricKey,$
$\quad LL2NVRAM.hashBarrier')$

The main precondition for the *MACUnforgeable* property is that the generated *MAC* is validated. We first prove the validation, which follows from the refinement.

$\langle 6 \rangle 1.$ $ValidateMAC(symmetricKey, ll2HistoryStateBinding, LL2RAM.authenticator')$
BY $\langle 4 \rangle 4$ DEF $ll2HistoryStateBinding$, $ll2HistorySummaryHash$

We then prove the generation, which follows from the *LL2CorruptRAM* action.

$\langle 6 \rangle 2.$ $LL2RAM.authenticator' = GenerateMAC(ll2RandomSymmetricKey, ll2Hash)$
$\langle 7 \rangle 1.$ $LL2RAM' = ll2UntrustedStorage$
BY $\langle 3 \rangle 2$
$\langle 7 \rangle 2.$ $ll2UntrustedStorage.authenticator = GenerateMAC(ll2RandomSymmetricKey, ll2Hash)$
BY $\langle 3 \rangle 2$
$\langle 7 \rangle 3.$ QED
BY $\langle 7 \rangle 1$, $\langle 7 \rangle 2$

The remaining preconditions are types.

$\langle 6 \rangle 3.$ $symmetricKey \in SymmetricKeyType$
BY $\langle 4 \rangle 3$
$\langle 6 \rangle 4.$ $ll2RandomSymmetricKey \in SymmetricKeyType$
$\langle 7 \rangle 1.$ $ll2RandomSymmetricKey \in SymmetricKeyType \setminus \{LL2NVRAM.symmetricKey\}$
BY $\langle 3 \rangle 2$
$\langle 7 \rangle 2.$ QED
BY $\langle 7 \rangle 1$
$\langle 6 \rangle 5.$ $ll2HistoryStateBinding \in HashType$
BY $\langle 4 \rangle 5$
$\langle 6 \rangle 6.$ $ll2Hash \in HashType$
BY $\langle 3 \rangle 2$

The *MACUnforgeable* property tells us that the two keys are equal.

$\langle 6 \rangle 7.$ $symmetricKey = ll2RandomSymmetricKey$
BY $\langle 6 \rangle 1$, $\langle 6 \rangle 2$, $\langle 6 \rangle 3$, $\langle 6 \rangle 4$, $\langle 6 \rangle 5$, $\langle 6 \rangle 6$, $MACUnforgeable$
$\langle 6 \rangle 8.$ QED
BY $\langle 4 \rangle 3$, $\langle 6 \rangle 7$

The third conjunct in the antecedent of the *AuthenticatorGeneratedLemma* mainly follows from the *LL2CorruptRAM* action, but we also have to prove that the history state binding specified in the refinement matches the arbitrary hash specified in the *LL2CorruptRAM* action.

$\langle 5 \rangle 10.$ $LL2RAM.authenticator' =$
$\quad GenerateMAC(ll2RandomSymmetricKey, ll2HistoryStateBinding)$

The state authenticaton in the primed Memoir-Opt RAM equals the authenticator specified by the existential *ll2UntrustedStorage* in the *LL2CorruptRAM* action.

$\langle 6 \rangle 1.$ $LL2RAM.authenticator' = ll2UntrustedStorage.authenticator$
$\langle 7 \rangle 1.$ $LL2RAM' = ll2UntrustedStorage$
BY $\langle 3 \rangle 2$
$\langle 7 \rangle 2.$ QED
BY $\langle 7 \rangle 1$

The authenticator specified by the existential $ll2UntrustedStorage$ in the $LL2CorruptRAM$ action is generated as a MAC of the history state binding from the $AuthenticatorsMatch$ predicate invoked by the refinement. This follows from the $MACCollisionResistant$ property.

$\langle 6 \rangle 2.$ $ll2UntrustedStorage.authenticator =$
$GenerateMAC(ll2RandomSymmetricKey, ll2HistoryStateBinding)$

The main precondition for the $MACUnforgeable$ property is that the generated MAC is validated. We first prove the validation, which follows from the refinement.

$\langle 7 \rangle 1.$ $ValidateMAC(symmetricKey, ll2HistoryStateBinding, LL2RAM.authenticator')$
BY $\langle 4 \rangle 4$ DEF $ll2HistoryStateBinding$, $ll2HistorySummaryHash$

We then prove the generation, which follows from the $LL2CorruptRAM$ action.

$\langle 7 \rangle 2.$ $LL2RAM.authenticator' = GenerateMAC(ll2RandomSymmetricKey, ll2Hash)$
$\langle 8 \rangle 1.$ $LL2RAM' = ll2UntrustedStorage$
BY $\langle 3 \rangle 2$
$\langle 8 \rangle 2.$ $ll2UntrustedStorage.authenticator = GenerateMAC(ll2RandomSymmetricKey, ll2Hash)$
BY $\langle 3 \rangle 2$
$\langle 8 \rangle 3.$ QED
BY $\langle 8 \rangle 1$, $\langle 8 \rangle 2$

The remaining preconditions are types.

$\langle 7 \rangle 3.$ $symmetricKey \in SymmetricKeyType$
BY $\langle 4 \rangle 3$
$\langle 7 \rangle 4.$ $ll2RandomSymmetricKey \in SymmetricKeyType$
$\langle 8 \rangle 1.$ $ll2RandomSymmetricKey \in SymmetricKeyType \setminus \{LL2NVRAM.symmetricKey\}$
BY $\langle 3 \rangle 2$
$\langle 8 \rangle 2.$ QED
BY $\langle 8 \rangle 1$
$\langle 7 \rangle 5.$ $ll2HistoryStateBinding \in HashType$
BY $\langle 4 \rangle 5$
$\langle 7 \rangle 6.$ $ll2Hash \in HashType$
BY $\langle 3 \rangle 2$

The $MACCollisionResistant$ property tells us that the two hash values are equal.

$\langle 7 \rangle 7.$ $ll2Hash = ll2HistoryStateBinding$
BY $\langle 7 \rangle 1$, $\langle 7 \rangle 2$, $\langle 7 \rangle 3$, $\langle 7 \rangle 4$, $\langle 7 \rangle 5$, $\langle 7 \rangle 6$, $MACCollisionResistant$
$\langle 7 \rangle 8.$ QED
BY $\langle 3 \rangle 2$, $\langle 7 \rangle 7$
$\langle 6 \rangle 3.$ QED
BY $\langle 6 \rangle 1$, $\langle 6 \rangle 2$

We then invoke the $AuthenticatorGeneratedLemma$ directly.

$\langle 5 \rangle 11.$ QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$, $\langle 5 \rangle 3$, $\langle 5 \rangle 4$, $\langle 5 \rangle 5$, $\langle 5 \rangle 6$, $\langle 5 \rangle 7$, $\langle 5 \rangle 8$, $\langle 5 \rangle 9$, $\langle 5 \rangle 10$,
$AuthenticatorGeneratedLemma$
DEF $ll1HistoryStateBinding$, $ll2HistorySummaryHash$, $ll2HistoryStateBinding$
$\langle 4 \rangle 10.$ QED
BY $\langle 4 \rangle 6$, $\langle 4 \rangle 7$, $\langle 4 \rangle 8$, $\langle 4 \rangle 9$

Since the refinement is an IF - THEN - ELSE , we prove the THEN and ELSE cases separately. For the THEN case, we assume that an extension is in progress and show that this refines to a $LL1RestrictedCorruption$ action.

$\langle 3 \rangle 4.$ ASSUME $LL2NVRAM.extensionInProgress$
PROVE $LL1RestrictedCorruption$

One fact that will be useful in several places below is that the extension field in the logical history summary of the Memoir-Opt $NVRAM$ and $SPCR$ is equal to a crazy hash value, because an extension is in progress but the $SPCR$ (in the primed state) equals the base hash value.

$\langle 4 \rangle 1.$ $LL2NVRAMLogicalHistorySummary.extension' = CrazyHashValue$

$\langle 5 \rangle 1.$ $LL2NVRAMLogicalHistorySummary' = [$
$anchor \mapsto LL2NVRAM.historySummaryAnchor',$
$extension \mapsto CrazyHashValue]$

$\langle 6 \rangle 1.$ $LL2NVRAM.extensionInProgress'$

$\langle 7 \rangle 1.$ $LL2NVRAM.extensionInProgress$
BY $\langle 3 \rangle 3$

$\langle 7 \rangle 2.$ UNCHANGED $LL2NVRAM.extensionInProgress$
BY $\langle 3 \rangle 2$

$\langle 7 \rangle 3.$ QED
BY $\langle 7 \rangle 1$, $\langle 7 \rangle 2$

$\langle 6 \rangle 2.$ $LL2SPCR' = BaseHashValue$
BY $\langle 3 \rangle 2$

$\langle 6 \rangle 3.$ QED
BY $\langle 6 \rangle 1$, $\langle 6 \rangle 2$ DEF $LL2NVRAMLogicalHistorySummary$

$\langle 5 \rangle 2.$ QED
BY $\langle 5 \rangle 1$

We prove each conjunct of $LL1RestrictedCorruption$ separately. First, we prove the conjunct relating to the $NVRAM$.

$\langle 4 \rangle 2.$ $LL1RestrictedCorruption!nvram$

The primed value of the history summary field in the Memoir-Basic $NVRAM$ serves as our witness for the garbage history summary.

$\langle 5 \rangle 1.$ $LL1NVRAM.historySummary' \in HashType$
BY $\langle 2 \rangle 1$ DEF $LL2Refinement$, $LL1TrustedStorageType$

We prove that the constraint labeled *current* in the $LL1RestrictedCorruption$ action is satisfied.

$\langle 5 \rangle 2.$ $LL1RestrictedCorruption!nvram!current(LL1NVRAM.historySummary')$

To prove the universally quantified expression, we take a set of variables of the appropriate types.

$\langle 6 \rangle 1.$ TAKE $stateHash1 \in HashType,$
$ll1Authenticator \in LL1ObservedAuthenticators$

We re-state the definition from within the $LL1RestrictedCorruption!nvram!current$ clause.

$\langle 6 \rangle$ $ll1GarbageHistoryStateBinding \triangleq Hash(LL1NVRAM.historySummary', stateHash1)$

We hide the definition.

$\langle 6 \rangle$ HIDE DEF $ll1GarbageHistoryStateBinding$

We need to prove the *nvram*:: *current* conjunct, which asserts that the authenticator is not a valid MAC for the history state binding formed from the history summary in the $NVRAM$ and any state hash.

$\langle 6 \rangle 2.$ $\neg ValidateMAC($
$LL1NVRAM.symmetricKey,$
$ll1GarbageHistoryStateBinding,$
$ll1Authenticator)$

We will use proof by contradiction.

$\langle 7 \rangle 1.$ SUFFICES
ASSUME
$ValidateMAC($
$LL1NVRAM.symmetricKey,$
$ll1GarbageHistoryStateBinding,$
$ll1Authenticator)$
PROVE
FALSE
OBVIOUS

We first pick, from the set of Memoir-Opt observed authenticators, a Memoir-Opt authenticator that matches the Memoir-Basic authenticator. We know that such a authenticator exists, because the refinement asserts that the sets of observed authenticators match across the two specs.

$\langle 7\rangle 2.$ PICK $ll2Authenticator \in LL2ObservedAuthenticators$:
$AuthenticatorsMatch($
$ll1Authenticator,$
$ll2Authenticator,$
$LL2NVRAM.symmetricKey,$
$LL2NVRAM.hashBarrier)$

$\langle 8\rangle 1.$ $ll1Authenticator \in LL1ObservedAuthenticators$
BY $\langle 6\rangle 1$

$\langle 8\rangle 2.$ $AuthenticatorSetsMatch($
$LL1ObservedAuthenticators,$
$LL2ObservedAuthenticators,$
$LL2NVRAM.symmetricKey,$
$LL2NVRAM.hashBarrier)$
BY $\langle 2\rangle 1$ DEF $LL2Refinement$

$\langle 8\rangle 3.$ QED
BY $\langle 8\rangle 1$, $\langle 8\rangle 2$ DEF $AuthenticatorSetsMatch$

We pick a set of variables of the appropriate types that satisfy the quantified *AuthenticatorsMatch* predicate.

$\langle 7\rangle 3.$ PICK $stateHash2 \in HashType,$
$ll1HistorySummary \in HashType,$
$ll2HistorySummary \in HistorySummaryType$:
$AuthenticatorsMatch($
$ll1Authenticator,$
$ll2Authenticator,$
$LL2NVRAM.symmetricKey,$
$LL2NVRAM.hashBarrier)!($
$stateHash2, ll1HistorySummary, ll2HistorySummary)!1$
BY $\langle 7\rangle 2$ DEF $AuthenticatorsMatch$

We re-state the definitions from the LET in *AuthenticatorsMatch*.

$\langle 7\rangle$ $ll1HistoryStateBinding \triangleq Hash(ll1HistorySummary, stateHash2)$
$\langle 7\rangle$ $ll2HistorySummaryHash \triangleq Hash(ll2HistorySummary.anchor, ll2HistorySummary.extension)$
$\langle 7\rangle$ $ll2HistoryStateBinding \triangleq Hash(ll2HistorySummaryHash, stateHash2)$

We prove the types of the definitions, with help from the *AuthenticatorsMatchDefsTypeSafeLemma*.

$\langle 7\rangle 4.$ $\wedge$ $ll1HistoryStateBinding \in HashType$
$\wedge$ $ll2HistorySummaryHash \in HashType$
$\wedge$ $ll2HistoryStateBinding \in HashType$
BY $\langle 7\rangle 3$, $AuthenticatorsMatchDefsTypeSafeLemma$

We hide the definitions.

$\langle 7\rangle$ HIDE DEF $ll1HistoryStateBinding, ll2HistorySummaryHash, ll2HistoryStateBinding$

We prove that the Memoir-Basic s history summary picked to satisfy the *AuthenticatorsMatch* predicate equals the history summary in the primed state of the Memoir-Basic *NVRAM*.

$\langle 7\rangle 5.$ $ll1HistorySummary = LL1NVRAM.historySummary'$

The first step is to show the equality of the history state bindings that bind each of these history summaries to their respective state hashes.

$\langle 8\rangle 1.$ $ll1HistoryStateBinding = ll1GarbageHistoryStateBinding$

By hypothesis, the authenticator is a valid *MAC* for the garbage history state binding.

$\langle 9\rangle 1.$ $ValidateMAC($
$LL2NVRAM.symmetricKey, ll1GarbageHistoryStateBinding, ll1Authenticator)$

$\langle 10\rangle 1.$ $LL1NVRAM.symmetricKey = LL2NVRAM.symmetricKey$
BY $\langle 2\rangle 1$ DEF $LL2Refinement$

$\langle 10\rangle 2.$ QED

BY ⟨7⟩1, ⟨10⟩1

The definition of the *AuthenticatorsMatch* predicate tells us that the Memoir-Basic authenticator was generated as a *MAC* from the history state binding.

⟨9⟩2. $ll1Authenticator = GenerateMAC(LL2NVRAM.symmetricKey, ll1HistoryStateBinding)$
BY ⟨7⟩3 DEF *ll1HistoryStateBinding*

The remaining preconditions are types.

⟨9⟩3. $LL2NVRAM.symmetricKey \in SymmetricKeyType$
BY ⟨2⟩1, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*
⟨9⟩4. $ll1HistoryStateBinding \in HashType$
BY ⟨7⟩4
⟨9⟩5. $ll1GarbageHistoryStateBinding \in HashType$
⟨10⟩1. $LL1NVRAM.historySummary' \in HashDomain$
BY ⟨5⟩1 DEF *HashDomain*
⟨10⟩2. $stateHash1 \in HashDomain$
BY ⟨5⟩1 DEF *HashDomain*
⟨10⟩3. QED
BY ⟨10⟩1, ⟨10⟩2, *HashTypeSafe*DEF *ll1GarbageHistoryStateBinding*

The *MACCollisionResistant* property tells us that the two history state bindings are equal.

⟨9⟩6. QED
BY ⟨9⟩1, ⟨9⟩2, ⟨9⟩3, ⟨9⟩4, ⟨9⟩5, *MACCollisionResistant*

By the collision resistance of the hash function, the equality of the history state bindings implies the equality of the history summaries.

⟨8⟩2. QED
⟨9⟩1. $LL1NVRAM.historySummary' \in HashDomain$
⟨10⟩1. $LL1NVRAM.historySummary' \in HashType$
BY ⟨2⟩1 DEF *LL2Refinement*, *LL1TrustedStorageType*
⟨10⟩2. QED
BY ⟨10⟩1 DEF *HashDomain*
⟨9⟩2. $ll1HistorySummary \in HashDomain$
⟨10⟩1. $ll1HistorySummary \in HashType$
BY ⟨7⟩3
⟨10⟩2. QED
BY ⟨10⟩1 DEF *HashDomain*
⟨9⟩3. $stateHash1 \in HashDomain$
⟨10⟩1. $stateHash1 \in HashType$
BY ⟨6⟩1
⟨10⟩2. QED
BY ⟨10⟩1 DEF *HashDomain*
⟨9⟩4. $stateHash2 \in HashDomain$
⟨10⟩1. $stateHash2 \in HashType$
BY ⟨7⟩3
⟨10⟩2. QED
BY ⟨10⟩1 DEF *HashDomain*
⟨9⟩5. QED
BY ⟨8⟩1, ⟨9⟩2, ⟨9⟩1, ⟨9⟩4, ⟨9⟩3, *HashCollisionResistant*
DEF *ll1HistoryStateBinding*, *ll1GarbageHistoryStateBinding*

The Memoir-Basic s history summary picked to satisfy the *AuthenticatorsMatch* predicate matches the primed logical history summary in the Memoir-Opt *NVRAM* and *SPCR*, by the refinement and the above equality.

⟨7⟩6. *HistorySummariesMatch*(
ll1HistorySummary,

$LL2NVRAMLogicalHistorySummary'$,
$LL2NVRAM.hashBarrier')$

$\langle 8\rangle 1.$ $HistorySummariesMatch($
$LL1NVRAM.historySummary'$,
$LL2NVRAMLogicalHistorySummary'$,
$LL2NVRAM.hashBarrier')$

BY $\langle 2\rangle 1$ DEF $LL2Refinement$

$\langle 8\rangle 2.$ QED

BY $\langle 7\rangle 5$, $\langle 8\rangle 1$

We prove that the *HistorySummariesMatch* predicate equals the *HistorySummariesMatchRecursion* predicate in this case. We assert each condition required by the definition of the predicate.

$\langle 7\rangle 7.$ $HistorySummariesMatch($
$ll1HistorySummary, LL2NVRAMLogicalHistorySummary', LL2NVRAM.hashBarrier') =$
$HistorySummariesMatchRecursion($
$ll1HistorySummary, LL2NVRAMLogicalHistorySummary', LL2NVRAM.hashBarrier')$

We prove some types, to satisfy the universal quantifiers in *HistorySummariesMatchDefinition*.

$\langle 8\rangle 1.$ $ll1HistorySummary \in HashType$

BY $\langle 7\rangle 3$

$\langle 8\rangle 2.$ $LL2NVRAMLogicalHistorySummary' \in HistorySummaryType$

BY $\langle 2\rangle 1$, $LL2NVRAMLogicalHistorySummaryTypeSafe$

$\langle 8\rangle 3.$ $LL2NVRAM.hashBarrier' \in HashType$

BY $\langle 2\rangle 1$, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$

We prove that the Memoir-Opt logical history summary does not equal the initial history summary.

$\langle 8\rangle$ $ll2InitialHistorySummary \triangleq [anchor \mapsto BaseHashValue, extension \mapsto BaseHashValue]$

$\langle 8\rangle 4.$ $LL2NVRAMLogicalHistorySummary' \neq ll2InitialHistorySummary$

$\langle 9\rangle 1.$ $LL2NVRAMLogicalHistorySummary.extension' \neq BaseHashValue$

BY $\langle 4\rangle 1$, $CrazyHashValueUnique$

$\langle 9\rangle 2.$ $ll2InitialHistorySummary.extension = BaseHashValue$

BY DEF $ll2InitialHistorySummary$

$\langle 9\rangle 3.$ QED

BY $\langle 9\rangle 1$, $\langle 9\rangle 2$

Finally, from *HistorySummariesMatchDefinition*, we can conclude that the *HistorySummariesMatch* predicate equals the quantified *HistorySummariesMatchRecursion* predicate.

$\langle 8\rangle 5.$ QED

BY $\langle 8\rangle 1$, $\langle 8\rangle 2$, $\langle 8\rangle 3$, $\langle 8\rangle 4$, $HistorySummariesMatchDefinition$
DEF $ll2InitialHistorySummary$

We pick values for the existential variables inside the *HistorySummariesMatchRecursion* predicate that satisfy the predicate. We know such variables exist, because the predicate is satisfied by the two previous steps.

$\langle 7\rangle 8.$ PICK $prevInput \in InputType$,
$previousLL1HistorySummary \in HashType$,
$previousLL2HistorySummary \in HistorySummaryType$:
$HistorySummariesMatchRecursion($
$ll1HistorySummary$,
$LL2NVRAMLogicalHistorySummary'$,
$LL2NVRAM.hashBarrier')!($
$prevInput$,
$previousLL1HistorySummary$,
$previousLL2HistorySummary)$

We prove some types, to satisfy the universal quantifiers in *HistorySummariesMatchRecursion* predicate.

$\langle 8\rangle 1.$ $ll1HistorySummary \in HashType$

BY $\langle 7 \rangle 3$

$\langle 8 \rangle 2$. $LL2NVRAMLogicalHistorySummary' \in HistorySummaryType$

BY $\langle 2 \rangle 1$, $LL2NVRAMLogicalHistorySummaryTypeSafe$

$\langle 8 \rangle 3$. $LL2NVRAM.hashBarrier' \in HashType$

BY $\langle 2 \rangle 1$, $LL2SubtypeImplicationLemma$ DEF $LL2SubtypeImplication$

$\langle 8 \rangle 4$. $HistorySummariesMatchRecursion($
$ll1HistorySummary, LL2NVRAMLogicalHistorySummary', LL2NVRAM.hashBarrier')$

BY $\langle 7 \rangle 6$, $\langle 7 \rangle 7$

$\langle 8 \rangle 5$. QED

BY $\langle 8 \rangle 1$, $\langle 8 \rangle 2$, $\langle 8 \rangle 4$ DEF $HistorySummariesMatchRecursion$

One of the conjuncts in the definition of *HistorySummariesMatchRecursion* is that the Memoir-Opt history summary is a successor of a previous history summary.

$\langle 7 \rangle 9$. $LL2HistorySummaryIsSuccessor($
$LL2NVRAMLogicalHistorySummary',$
$previousLL2HistorySummary,$
$prevInput,$
$LL2NVRAM.hashBarrier')$

BY $\langle 7 \rangle 8$ DEF $HistorySummariesMatchRecursion$

We re-state the definitions from the LET in *LL2HistorySummaryIsSuccessor*.

$\langle 7 \rangle$ $successorHistorySummary \triangleq$
$Successor(previousLL2HistorySummary, prevInput, LL2NVRAM.hashBarrier')$

$\langle 7 \rangle$ $checkpointedSuccessorHistorySummary \triangleq Checkpoint(successorHistorySummary)$

We hide the definitions.

$\langle 7 \rangle$ HIDE DEF $successorHistorySummary$, $checkpointedSuccessorHistorySummary$

The definition of *LL2HistorySummaryIsSuccessor* tells us that there are two ways that the logical history summary could be a successor. We will prove that neither of these disjuncts is satisfiable.

$\langle 7 \rangle 10$. $\lor$ $LL2NVRAMLogicalHistorySummary' = successorHistorySummary$
$\lor$ $LL2NVRAMLogicalHistorySummary' = checkpointedSuccessorHistorySummary$

BY $\langle 7 \rangle 9$

DEF $LL2HistorySummaryIsSuccessor$, $successorHistorySummary$,
$checkpointedSuccessorHistorySummary$

First, we prove that the logical history summary cannot be a successor.

$\langle 7 \rangle 11$. $LL2NVRAMLogicalHistorySummary' \neq successorHistorySummary$

We re-state a definition from the LET in the *Successor* operator.

$\langle 8 \rangle$ $securedInput \triangleq Hash(LL2NVRAM.hashBarrier', prevInput)$

We hide the definition.

$\langle 8 \rangle$ HIDE DEF $securedInput$

There is only one sub-step, which is proving that the extension fields of these two records are unequal.

$\langle 8 \rangle 1$. $LL2NVRAMLogicalHistorySummary.extension' \neq successorHistorySummary.extension$

If an extension is in progress but the *SPCR* equals the base hash value, the logical history summary equals a crazy hash value, as proven above.

$\langle 9 \rangle 1$. $LL2NVRAMLogicalHistorySummary.extension' = CrazyHashValue$

BY $\langle 4 \rangle 1$

The extension field of the successor history summary is equal to a hash generated by the hash function.

$\langle 9 \rangle 2$. $successorHistorySummary.extension =$
$Hash(previousLL2HistorySummary.extension, securedInput)$

BY DEF $successorHistorySummary$, $Successor$, $securedInput$

The arguments to the hash function are both in the hash domain.

$\langle 9 \rangle 3$. $previousLL2HistorySummary.extension \in HashDomain$

$\langle 10\rangle 1.$ *previousLL2HistorySummary.extension* $\in$ *HashType*
$\langle 11\rangle 1.$ *previousLL2HistorySummary* $\in$ *HistorySummaryType*
BY $\langle 7\rangle 8$
$\langle 11\rangle 2.$ QED
BY $\langle 11\rangle 1$ DEF *HistorySummaryType*
$\langle 10\rangle 2.$ QED
BY $\langle 10\rangle 1$ DEF *HashDomain*
$\langle 9\rangle 4.$ *securedInput* $\in$ *HashDomain*
$\langle 10\rangle 1.$ *securedInput* $\in$ *HashType*
$\langle 11\rangle 1.$ *LL2NVRAM.hashBarrier'* $\in$ *HashDomain*
$\langle 12\rangle 1.$ *LL2NVRAM.hashBarrier'* $\in$ *HashType*
BY $\langle 2\rangle 1$, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*
$\langle 12\rangle 2.$ QED
BY $\langle 12\rangle 1$ DEF *HashDomain*
$\langle 11\rangle 2.$ *prevInput* $\in$ *HashDomain*
$\langle 12\rangle 1.$ *prevInput* $\in$ *InputType*
BY $\langle 7\rangle 8$
$\langle 12\rangle 2.$ QED
BY $\langle 12\rangle 1$ DEF *HashDomain*
$\langle 11\rangle 3.$ QED
BY $\langle 11\rangle 1$, $\langle 11\rangle 2$, *HashTypeSafe*DEF *securedInput*
$\langle 10\rangle 2.$ QED
BY $\langle 10\rangle 1$ DEF *HashDomain*

The crazy hash value is not equal to any hash value that can be generated by the hash function when operating on arguments within its domain.

$\langle 9\rangle 5.$ QED
BY $\langle 9\rangle 1$, $\langle 9\rangle 2$, $\langle 9\rangle 3$, $\langle 9\rangle 4$, *CrazyHashValueUnique*
$\langle 8\rangle 2.$ QED
BY $\langle 8\rangle 1$

Second, we prove that the logical history summary cannot be a checkpoint.

$\langle 7\rangle 12.$ *LL2NVRAMLogicalHistorySummary'* $\neq$ *checkpointedSuccessorHistorySummary*

The extension field of the logical history summary does not equal the base hash value, as we proved above.

$\langle 8\rangle 1.$ *LL2NVRAMLogicalHistorySummary.extension'* $\neq$ *BaseHashValue*
BY $\langle 4\rangle 1$, *CrazyHashValueUnique*

The extension field of the checkpointed successor does equal the base hash value. This follows from the *CheckpointHasBaseExtensionLemma*.

$\langle 8\rangle 2.$ *checkpointedSuccessorHistorySummary.extension* $=$ *BaseHashValue*
$\langle 9\rangle 1.$ *successorHistorySummary* $\in$ *HistorySummaryType*
$\langle 10\rangle 1.$ *previousLL2HistorySummary* $\in$ *HistorySummaryType*
BY $\langle 7\rangle 8$
$\langle 10\rangle 2.$ *prevInput* $\in$ *InputType*
BY $\langle 7\rangle 8$
$\langle 10\rangle 3.$ *LL2NVRAM.hashBarrier'* $\in$ *HashType*
BY $\langle 2\rangle 1$, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*
$\langle 10\rangle 4.$ QED
BY $\langle 10\rangle 1$, $\langle 10\rangle 2$, $\langle 10\rangle 3$, *SuccessorTypeSafe*DEF *successorHistorySummary*
$\langle 9\rangle 2.$ QED
BY $\langle 9\rangle 1$, *CheckpointHasBaseExtensionLemma*
DEF *checkpointedSuccessorHistorySummary*
$\langle 8\rangle 3.$ QED
BY $\langle 8\rangle 1$, $\langle 8\rangle 2$

We thus have a contradiction.

⟨7⟩13. QED

BY ⟨7⟩10, ⟨7⟩11, ⟨7⟩12

⟨6⟩3. QED

BY ⟨6⟩2 DEF $ll1GarbageHistoryStateBinding$

We prove that the constraint labeled *previous* in the $LL1RestrictedCorruption$ action is satisfied.

⟨5⟩3. $LL1RestrictedCorruption!nvram!previous(LL1NVRAM.historySummary')$

To prove the universally quantified expression, we take a set of variables of the appropriate types.

⟨6⟩1. TAKE $stateHash1 \in HashType$,
$ll1Authenticator \in LL1ObservedAuthenticators$,
$ll1SomeHistorySummary \in HashType$,
$someInput \in InputType$

We re-state the definition from within the $LL1RestrictedCorruption!nvram!previous$ clause.

⟨6⟩ $ll1SomeHistoryStateBinding \triangleq Hash(ll1SomeHistorySummary, stateHash1)$

We hide the definition.

⟨6⟩ HIDE DEF $ll1SomeHistoryStateBinding$

We need to prove the *nvram*:: *previous* conjunct, which asserts an implication. It suffices to assume the antecedent and prove the consequent.

⟨6⟩2. SUFFICES
ASSUME $LL1NVRAM.historySummary' = Hash(ll1SomeHistorySummary, someInput)$
PROVE $\neg ValidateMAC($
$LL1NVRAM.symmetricKey,$
$ll1SomeHistoryStateBinding,$
$ll1Authenticator)$

BY DEF $ll1SomeHistoryStateBinding$

The consequent of the *nvram*:: *previous* conjunct asserts that the authenticator is not a valid MAC for the history state binding formed from any predecessor of the history summary in the $NVRAM$ and any state hash.

⟨6⟩3. $\neg ValidateMAC($
$LL1NVRAM.symmetricKey,$
$ll1SomeHistoryStateBinding,$
$ll1Authenticator)$

We will use proof by contradiction.

⟨7⟩1. SUFFICES
ASSUME
$ValidateMAC($
$LL1NVRAM.symmetricKey,$
$ll1SomeHistoryStateBinding,$
$ll1Authenticator)$
PROVE
FALSE

OBVIOUS

We first pick, from the set of Memoir-Opt observed authenticators, a Memoir-Opt authenticator that matches the Memoir-Basic authenticator. We know that such a authenticator exists, because the refinement asserts that the sets of observed authenticators match across the two specs.

⟨7⟩2. PICK $ll2Authenticator \in LL2ObservedAuthenticators$:
$AuthenticatorsMatch($
$ll1Authenticator,$
$ll2Authenticator,$
$LL2NVRAM.symmetricKey,$

$LL2NVRAM.hashBarrier)$

⟨8⟩1. $ll1Authenticator \in LL1ObservedAuthenticators$

BY ⟨6⟩1

⟨8⟩2. $AuthenticatorSetsMatch($
$LL1ObservedAuthenticators,$
$LL2ObservedAuthenticators,$
$LL2NVRAM.symmetricKey,$
$LL2NVRAM.hashBarrier)$

BY ⟨2⟩1 DEF $LL2Refinement$

⟨8⟩3. QED

BY ⟨8⟩1, ⟨8⟩2 DEF $AuthenticatorSetsMatch$

We pick a set of variables of the appropriate types that satisfy the quantified $AuthenticatorsMatch$ predicate.

⟨7⟩3. PICK $stateHash2 \in HashType,$
$ll1HistorySummary \in HashType,$
$ll2HistorySummary \in HistorySummaryType :$
$AuthenticatorsMatch($
$ll1Authenticator,$
$ll2Authenticator,$
$LL2NVRAM.symmetricKey,$
$LL2NVRAM.hashBarrier)!($
$stateHash2, ll1HistorySummary, ll2HistorySummary)!1$

BY ⟨7⟩2 DEF $AuthenticatorsMatch$

We re-state the definitions from the LET in $AuthenticatorsMatch$.

⟨7⟩ $ll1HistoryStateBinding \triangleq Hash(ll1HistorySummary, stateHash2)$

⟨7⟩ $ll2HistorySummaryHash \triangleq Hash(ll2HistorySummary.anchor, ll2HistorySummary.extension)$

⟨7⟩ $ll2HistoryStateBinding \triangleq Hash(ll2HistorySummaryHash, stateHash2)$

We prove the types of the definitions, with help from the $AuthenticatorsMatchDefsTypeSafeLemma$.

⟨7⟩4. $\land\ ll1HistoryStateBinding \in HashType$
$\land\ ll2HistorySummaryHash \in HashType$
$\land\ ll2HistoryStateBinding \in HashType$

BY ⟨7⟩3, $AuthenticatorsMatchDefsTypeSafeLemma$

We hide the definitions.

⟨7⟩ HIDE DEF $ll1HistoryStateBinding, ll2HistorySummaryHash, ll2HistoryStateBinding$

We prove that the Memoir-Basic s history summary picked to satisfy the $AuthenticatorsMatch$ predicate equals the history summary in the primed state of the Memoir-Basic $NVRAM$.

⟨7⟩5. $ll1HistorySummary = ll1SomeHistorySummary$

The first step is to show the equality of the history state bindings that bind each of these history summaries to their respective state hashes.

⟨8⟩1. $ll1HistoryStateBinding = ll1SomeHistoryStateBinding$

By hypothesis, the authenticator is a valid MAC for the garbage history state binding.

⟨9⟩1. $ValidateMAC($
$LL2NVRAM.symmetricKey, ll1SomeHistoryStateBinding, ll1Authenticator)$

⟨10⟩1. $LL1NVRAM.symmetricKey = LL2NVRAM.symmetricKey$

BY ⟨2⟩1 DEF $LL2Refinement$

⟨10⟩2. QED

BY ⟨7⟩1, ⟨10⟩1

The definition of the $AuthenticatorsMatch$ predicate tells us that the Memoir-Basic authenticator was generated as a MAC from the history state binding.

⟨9⟩2. $ll1Authenticator = GenerateMAC(LL2NVRAM.symmetricKey, ll1HistoryStateBinding)$

BY ⟨7⟩3 DEF $ll1HistoryStateBinding$

The remaining preconditions are types.

$\langle 9\rangle 3.$ $LL2NVRAM.symmetricKey \in SymmetricKeyType$

BY $\langle 2\rangle 1$, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$

$\langle 9\rangle 4.$ $ll1HistoryStateBinding \in HashType$

BY $\langle 7\rangle 4$

$\langle 9\rangle 5.$ $ll1SomeHistoryStateBinding \in HashType$

$\langle 10\rangle 1.$ $ll1SomeHistorySummary \in HashDomain$

$\langle 11\rangle 1.$ $ll1SomeHistorySummary \in HashType$

BY $\langle 6\rangle 1$

$\langle 11\rangle 2.$ QED

BY $\langle 11\rangle 1$ DEF $HashDomain$

$\langle 10\rangle 2.$ $stateHash1 \in HashDomain$

BY $\langle 5\rangle 1$ DEF $HashDomain$

$\langle 10\rangle 3.$ QED

BY $\langle 10\rangle 1$, $\langle 10\rangle 2$, $HashTypeSafe$DEF $ll1SomeHistoryStateBinding$

The $MACCollisionResistant$ property tells us that the two history state bindings are equal.

$\langle 9\rangle 6.$ QED

BY $\langle 9\rangle 1$, $\langle 9\rangle 2$, $\langle 9\rangle 3$, $\langle 9\rangle 4$, $\langle 9\rangle 5$, $MACCollisionResistant$

By the collision resistance of the hash function, the equality of the history state bindings implies the equality of the history summaries.

$\langle 8\rangle 2.$ QED

$\langle 9\rangle 1.$ $ll1SomeHistorySummary \in HashDomain$

$\langle 10\rangle 1.$ $ll1SomeHistorySummary \in HashType$

BY $\langle 6\rangle 1$

$\langle 10\rangle 2.$ QED

BY $\langle 10\rangle 1$ DEF $HashDomain$

$\langle 9\rangle 2.$ $ll1HistorySummary \in HashDomain$

$\langle 10\rangle 1.$ $ll1HistorySummary \in HashType$

BY $\langle 7\rangle 3$

$\langle 10\rangle 2.$ QED

BY $\langle 10\rangle 1$ DEF $HashDomain$

$\langle 9\rangle 3.$ $stateHash1 \in HashDomain$

$\langle 10\rangle 1.$ $stateHash1 \in HashType$

BY $\langle 6\rangle 1$

$\langle 10\rangle 2.$ QED

BY $\langle 10\rangle 1$ DEF $HashDomain$

$\langle 9\rangle 4.$ $stateHash2 \in HashDomain$

$\langle 10\rangle 1.$ $stateHash2 \in HashType$

BY $\langle 7\rangle 3$

$\langle 10\rangle 2.$ QED

BY $\langle 10\rangle 1$ DEF $HashDomain$

$\langle 9\rangle 5.$ QED

BY $\langle 8\rangle 1$, $\langle 9\rangle 2$, $\langle 9\rangle 1$, $\langle 9\rangle 4$, $\langle 9\rangle 3$, $HashCollisionResistant$

DEF $ll1HistoryStateBinding$, $ll1SomeHistoryStateBinding$

We pick a value for the Memoir-Opt previous-inputs-summary existential variable inside the *HistorySummariesMatchRecursion* predicate that satisfies this predicate for (1) the Memoir-Basic s history summary picked to satisfy the *AuthenticatorsMatch* predicate and (2) the input taken from the universal quantifier in the *previous* conjunct in the *LL1RestrictedCorruption* action.

$\langle 7\rangle 6.$ PICK $previousLL2HistorySummary \in HistorySummaryType$:

$HistorySummariesMatchRecursion($

$LL1NVRAM.historySummary'$,

$LL2NVRAMLogicalHistorySummary'$,

$LL2NVRAM.hashBarrier')!($
$someInput,$
$ll1SomeHistorySummary,$
$previousLL2HistorySummary)$

The Memoir-Basic s history summary picked to satisfy the *AuthenticatorsMatch* predicate matches the primed logical history summary in the Memoir-Opt *NVRAM* and *SPCR*, by the refinement and the above equality.

$\langle 8\rangle 1.$ $HistorySummariesMatch($
$LL1NVRAM.historySummary',$
$LL2NVRAMLogicalHistorySummary',$
$LL2NVRAM.hashBarrier')$

$\langle 9\rangle 1.$ $HistorySummariesMatch($
$LL1NVRAM.historySummary',$
$LL2NVRAMLogicalHistorySummary',$
$LL2NVRAM.hashBarrier')$
BY $\langle 2\rangle 1$ DEF $LL2Refinement$

$\langle 9\rangle 2.$ QED
BY $\langle 9\rangle 1$

We prove that the *HistorySummariesMatch* predicate equals the *HistorySummariesMatchRecursion* predicate in this case. We assert each condition required by the definition of the predicate.

$\langle 8\rangle 2.$ $HistorySummariesMatch($
$LL1NVRAM.historySummary',$
$LL2NVRAMLogicalHistorySummary',$
$LL2NVRAM.hashBarrier') =$
$HistorySummariesMatchRecursion($
$LL1NVRAM.historySummary',$
$LL2NVRAMLogicalHistorySummary',$
$LL2NVRAM.hashBarrier')$

We prove some types, to satisfy the universal quantifiers in *HistorySummariesMatchDefinition*.

$\langle 9\rangle 1.$ $LL1NVRAM.historySummary' \in HashType$
BY $\langle 2\rangle 1$ DEF $LL2Refinement$, $LL1TrustedStorageType$

$\langle 9\rangle 2.$ $LL2NVRAMLogicalHistorySummary' \in HistorySummaryType$
BY $\langle 2\rangle 1$, $LL2NVRAMLogicalHistorySummaryTypeSafe$

$\langle 9\rangle 3.$ $LL2NVRAM.hashBarrier' \in HashType$
BY $\langle 2\rangle 1$, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$

We prove that the Memoir-Opt logical history summary does not equal the initial history summary.

$\langle 9\rangle$ $ll2InitialHistorySummary \triangleq [anchor \mapsto BaseHashValue, extension \mapsto BaseHashValue]$

$\langle 9\rangle 4.$ $LL2NVRAMLogicalHistorySummary' \neq ll2InitialHistorySummary$

$\langle 10\rangle 1.$ $LL2NVRAMLogicalHistorySummary.extension' \neq BaseHashValue$
BY $\langle 4\rangle 1$, $CrazyHashValueUnique$

$\langle 10\rangle 2.$ $ll2InitialHistorySummary.extension = BaseHashValue$
BY DEF $ll2InitialHistorySummary$

$\langle 10\rangle 3.$ QED
BY $\langle 10\rangle 1$, $\langle 10\rangle 2$

Finally, from *HistorySummariesMatchDefinition*, we can conclude that the *HistorySummariesMatch* predicate equals the quantified *HistorySummariesMatchRecursion* predicate.

$\langle 9\rangle 5.$ QED
BY $\langle 9\rangle 1$, $\langle 9\rangle 2$, $\langle 9\rangle 3$, $\langle 9\rangle 4$, $HistorySummariesMatchDefinition$
DEF $ll2InitialHistorySummary$

We pick values for the remaining two existential variables inside the *HistorySummariesMatchRecursion* predicate that satisfy the predicate. We know such variables exist, because the predicate is satisfied by the two previous steps.

⟨8⟩3. PICK $prevInput \in InputType$,
$previousLL1HistorySummary \in HashType$,
$previousLL2HistorySummary \in HistorySummaryType$:
$HistorySummariesMatchRecursion($
$LL1NVRAM.historySummary'$,
$LL2NVRAMLogicalHistorySummary'$,
$LL2NVRAM.hashBarrier')!($
$prevInput$,
$previousLL1HistorySummary$,
$previousLL2HistorySummary)$

We prove some types, to satisfy the universal quantifiers in *HistorySummariesMatchRecursion* predicate.

⟨9⟩1. $ll1HistorySummary \in HashType$
BY ⟨7⟩3
⟨9⟩2. $LL2NVRAMLogicalHistorySummary' \in HistorySummaryType$
BY ⟨2⟩1, *LL2NVRAMLogicalHistorySummaryTypeSafe*
⟨9⟩3. $LL2NVRAM.hashBarrier' \in HashType$
BY ⟨2⟩1, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*
⟨9⟩4. $HistorySummariesMatchRecursion($
$LL1NVRAM.historySummary'$,
$LL2NVRAMLogicalHistorySummary'$,
$LL2NVRAM.hashBarrier')$
BY ⟨8⟩1, ⟨8⟩2
⟨9⟩5. QED
BY ⟨9⟩1, ⟨9⟩2, ⟨9⟩4 DEF *HistorySummariesMatchRecursion*

We prove that the existential variables for the previous input and the previous history summary in the above pick are equal to the input and history summary taken from the universal quantifiers in the *previous* conjunct in the *LL1RestrictedCorruption* action. We use the *HashCollisionResistant* property.

⟨8⟩4. $\wedge\ ll1SomeHistorySummary = previousLL1HistorySummary$
$\wedge\ someInput = prevInput$

We prove the necessary types for the *HashCollisionResistant* property.

⟨9⟩1. $ll1SomeHistorySummary \in HashDomain$
⟨10⟩1. $ll1SomeHistorySummary \in HashType$
BY ⟨6⟩1
⟨10⟩2. QED
BY ⟨10⟩1 DEF *HashDomain*
⟨9⟩2. $someInput \in HashDomain$
⟨10⟩1. $someInput \in InputType$
BY ⟨6⟩1
⟨10⟩2. QED
BY ⟨10⟩1 DEF *HashDomain*
⟨9⟩3. $previousLL1HistorySummary \in HashDomain$
⟨10⟩1. $previousLL1HistorySummary \in HashType$
BY ⟨8⟩3
⟨10⟩2. QED
BY ⟨10⟩1 DEF *HashDomain*
⟨9⟩4. $prevInput \in HashDomain$
⟨10⟩1. $prevInput \in InputType$
BY ⟨8⟩3
⟨10⟩2. QED
BY ⟨10⟩1 DEF *HashDomain*

The hashes are equal, because each is equal to the history summary in the primed Memoir-Basic *NVRAM*.

$\langle 9\rangle 5.$ $Hash(ll1SomeHistorySummary, someInput) =$
$Hash(previousLL1HistorySummary, prevInput)$

The hash of the taken history summary and input are equal to the history summary in the primed Memoir-Basic *NVRAM* by assumption of the antecedent in the *previous* conjunct in the *LL1RestrictedCorruption* action.

$\langle 10\rangle 1.$ $LL1NVRAM.historySummary' = Hash(ll1SomeHistorySummary, someInput)$
BY $\langle 6\rangle 2$

The hash of the picked history summary and input are equal to the history summary in the primed Memoir-Basic *NVRAM* by the definition of the *HistorySummariesMatchRecursion* predicate.

$\langle 10\rangle 2.$ $LL1NVRAM.historySummary' = Hash(previousLL1HistorySummary, prevInput)$
BY $\langle 8\rangle 3$

$\langle 10\rangle 3.$ QED
BY $\langle 10\rangle 1$, $\langle 10\rangle 2$

$\langle 9\rangle 6.$ QED
BY $\langle 9\rangle 1$, $\langle 9\rangle 2$, $\langle 9\rangle 3$, $\langle 9\rangle 4$, $\langle 9\rangle 5$, *HashCollisionResistant*

$\langle 8\rangle 5.$ QED
BY $\langle 8\rangle 3$, $\langle 8\rangle 4$

One of the conjuncts in the definition of *HistorySummariesMatchRecursion* is that the Memoir-Opt history summary is a successor of a previous history summary.

$\langle 7\rangle 7.$ $LL2HistorySummaryIsSuccessor($
$LL2NVRAMLogicalHistorySummary',$
$previousLL2HistorySummary,$
$someInput,$
$LL2NVRAM.hashBarrier')$
BY $\langle 7\rangle 6$ DEF *HistorySummariesMatchRecursion*

We re-state the definitions from the LET in *LL2HistorySummaryIsSuccessor*.

$\langle 7\rangle$ $successorHistorySummary \triangleq$
$Successor(previousLL2HistorySummary, someInput, LL2NVRAM.hashBarrier')$
$\langle 7\rangle$ $checkpointedSuccessorHistorySummary \triangleq Checkpoint(successorHistorySummary)$

We hide the definitions.

$\langle 7\rangle$ HIDE DEF *successorHistorySummary*, *checkpointedSuccessorHistorySummary*

The definition of *LL2HistorySummaryIsSuccessor* tells us that there are two ways that the logical history summary could be a successor. We will prove that neither of these disjuncts is satisfiable.

$\langle 7\rangle 8.$ $\vee$ $LL2NVRAMLogicalHistorySummary' = successorHistorySummary$
$\vee$ $LL2NVRAMLogicalHistorySummary' = checkpointedSuccessorHistorySummary$
BY $\langle 7\rangle 7$
DEF *LL2HistorySummaryIsSuccessor*, *successorHistorySummary*,
checkpointedSuccessorHistorySummary

First, we prove that the logical history summary cannot be a successor.

$\langle 7\rangle 9.$ $LL2NVRAMLogicalHistorySummary' \neq successorHistorySummary$

We re-state a definition from the LET in the *Successor* operator.

$\langle 8\rangle$ $securedInput \triangleq Hash(LL2NVRAM.hashBarrier', someInput)$

We hide the definition.

$\langle 8\rangle$ HIDE DEF *securedInput*

There is only one sub-step, which is proving that the extension fields of these two records are unequal.

$\langle 8\rangle 1.$ $LL2NVRAMLogicalHistorySummary.extension' \neq successorHistorySummary.extension$

The extension field of the logical history summary is a crazy hash value, as proven above.

$\langle 9\rangle 1.$ $LL2NVRAMLogicalHistorySummary.extension' = CrazyHashValue$
BY $\langle 4\rangle 1$

Second, we prove that the extension field of the successor history summary from *LL2HistorySummaryIsSuccessor* is computed from the hash function.

$\langle 9\rangle 2.$ $successorHistorySummary.extension =$
$\quad Hash(previousLL2HistorySummary.extension,\ securedInput)$
BY DEF *successorHistorySummary*, *Successor*, *securedInput*

Third, we prove that the two hashes are unequal. We will use the *CrazyHashValueUnique* property.

$\langle 9\rangle 3.$ $Hash(previousLL2HistorySummary.extension,\ securedInput) \neq CrazyHashValue$

To employ the *CrazyHashValueUnique* property, we need to prove some types.

$\langle 10\rangle 1.$ $previousLL2HistorySummary.extension \in HashDomain$
$\langle 11\rangle 1.$ $previousLL2HistorySummary.extension \in HashType$
$\langle 12\rangle 1.$ $previousLL2HistorySummary \in HistorySummaryType$
BY $\langle 7\rangle 6$
$\langle 12\rangle 2.$ QED
BY $\langle 12\rangle 1$ DEF *HistorySummaryType*
$\langle 11\rangle 2.$ QED
BY $\langle 11\rangle 1$ DEF *HashDomain*
$\langle 10\rangle 2.$ $securedInput \in HashDomain$
$\langle 11\rangle 1.$ $securedInput \in HashType$
$\langle 12\rangle 1.$ $LL2NVRAM.hashBarrier' \in HashDomain$
$\langle 13\rangle 1.$ $LL2NVRAM.hashBarrier' \in HashType$
BY $\langle 2\rangle 1$, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*
$\langle 13\rangle 2.$ QED
BY $\langle 13\rangle 1$ DEF *HashDomain*
$\langle 12\rangle 2.$ $someInput \in HashDomain$
$\langle 13\rangle 1.$ $someInput \in InputType$
BY $\langle 6\rangle 1$
$\langle 13\rangle 2.$ QED
BY $\langle 13\rangle 1$ DEF *HashDomain*
$\langle 12\rangle 3.$ QED
BY $\langle 12\rangle 1$, $\langle 12\rangle 2$, *HashTypeSafe*DEF *securedInput*
$\langle 11\rangle 2.$ QED
BY $\langle 11\rangle 1$ DEF *HashDomain*
$\langle 10\rangle 3.$ QED
BY $\langle 10\rangle 1$, $\langle 10\rangle 2$, *CrazyHashValueUnique*
$\langle 9\rangle 4.$ QED
BY $\langle 9\rangle 1$, $\langle 9\rangle 2$, $\langle 9\rangle 3$
$\langle 8\rangle 2.$ QED
BY $\langle 8\rangle 1$

Second, we prove that the logical history summary cannot be a checkpoint.

$\langle 7\rangle 10.$ $LL2NVRAMLogicalHistorySummary' \neq checkpointedSuccessorHistorySummary$

The extension field of the logical history summary does not equal the base hash value, as we proved above.

$\langle 8\rangle 1.$ $LL2NVRAMLogicalHistorySummary.extension' \neq BaseHashValue$
BY $\langle 4\rangle 1$, *CrazyHashValueUnique*

The extension field of the checkpointed successor does equal the base hash value. This follows from the *CheckpointHasBaseExtensionLemma*.

$\langle 8\rangle 2.$ $checkpointedSuccessorHistorySummary.extension = BaseHashValue$
$\langle 9\rangle 1.$ $successorHistorySummary \in HistorySummaryType$
$\langle 10\rangle 1.$ $previousLL2HistorySummary \in HistorySummaryType$
BY $\langle 7\rangle 6$
$\langle 10\rangle 2.$ $someInput \in InputType$
BY $\langle 6\rangle 1$

$\langle 10\rangle 3.$ $LL2NVRAM.hashBarrier' \in HashType$

BY $\langle 2\rangle 1$, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*

$\langle 10\rangle 4.$ QED

BY $\langle 10\rangle 1$, $\langle 10\rangle 2$, $\langle 10\rangle 3$, *SuccessorTypeSafe*DEF *successorHistorySummary*

$\langle 9\rangle 2.$ QED

BY $\langle 9\rangle 1$, *CheckpointHasBaseExtensionLemma*
DEF *checkpointedSuccessorHistorySummary*

$\langle 8\rangle 3.$ QED

BY $\langle 8\rangle 1$, $\langle 8\rangle 2$

We thus have a contradiction.

$\langle 7\rangle 11.$ QED

BY $\langle 7\rangle 8$, $\langle 7\rangle 9$, $\langle 7\rangle 10$

$\langle 6\rangle 4.$ QED

BY $\langle 6\rangle 3$ DEF *ll1SomeHistoryStateBinding*

We prove the third conjunct within the *nvram* conjunct of the *LL1RestrictedCorruption* action.

$\langle 5\rangle 4.$ $LL1NVRAM' = [$
$historySummary \mapsto LL1NVRAM.historySummary',$
$symmetricKey \mapsto LL1NVRAM.symmetricKey]$

$\langle 6\rangle 1.$ $LL1NVRAM \in LL1TrustedStorageType$

BY $\langle 2\rangle 1$ DEF *LL2Refinement*

$\langle 6\rangle 2.$ $LL1NVRAM' \in LL1TrustedStorageType$

BY $\langle 2\rangle 1$ DEF *LL2Refinement*

$\langle 6\rangle 3.$ UNCHANGED $LL1NVRAM.symmetricKey$

$\langle 7\rangle 1.$ UNCHANGED $LL2NVRAM.symmetricKey$

BY $\langle 3\rangle 2$

$\langle 7\rangle 2.$ QED

BY $\langle 2\rangle 1$, $\langle 7\rangle 1$, *UnchangedNVRAMSymmetricKeyLemma*

$\langle 6\rangle 4.$ QED

BY $\langle 6\rangle 1$, $\langle 6\rangle 2$, $\langle 6\rangle 3$, *LL1NVRAMRecordCompositionLemma*

$\langle 5\rangle 5.$ QED

BY $\langle 5\rangle 1$, $\langle 5\rangle 2$, $\langle 5\rangle 3$, $\langle 5\rangle 4$

The *ram* conjunct of the *LL1RestrictedCorruption* action is satisfied because the *trashed* disjunct is satisfied.

$\langle 4\rangle 3.$ $LL1RestrictedCorruption!ram$

The *trashed* disjunct of the *LL1RestrictedCorruption* action is satisfied by the proof above.

$\langle 5\rangle 1.$ $LL1RestrictedCorruption!ram!trashed$

BY $\langle 3\rangle 3$

$\langle 5\rangle 2.$ QED

BY $\langle 5\rangle 1$

The disk is unchanged by the *UnchangedDiskLemma*.

$\langle 4\rangle 4.$ UNCHANGED $LL1Disk$

BY $\langle 2\rangle 1$, $\langle 3\rangle 2$, *UnchangedDiskLemma*

The set of available inputs is unchanged by the *UnchangedAvailableInputsLemma*.

$\langle 4\rangle 5.$ UNCHANGED $LL1AvailableInputs$

BY $\langle 2\rangle 1$, $\langle 3\rangle 2$, *UnchangedAvailableInputsLemma*

The set of observed outputs is unchanged by the *UnchangedObservedOutputsLemma*.

$\langle 4\rangle 6.$ UNCHANGED $LL1ObservedOutputs$

BY $\langle 2\rangle 1$, $\langle 3\rangle 2$, *UnchangedObservedOutputsLemma*

The set of observed authenticators is unchanged by the *UnchangedObservedAuthenticatorsLemma*.

$\langle 4\rangle 7.$ UNCHANGED $LL1ObservedAuthenticators$

BY $\langle 2\rangle 1$, $\langle 3\rangle 2$, *UnchangedObservedAuthenticatorsLemma*

Lastly, we tie together all of the required aspects of the $LL1RestrictedCorruption$ definition.

$\langle 4 \rangle 8.$ QED
BY $\langle 4 \rangle 2$, $\langle 4 \rangle 3$, $\langle 4 \rangle 4$, $\langle 4 \rangle 5$, $\langle 4 \rangle 6$, $\langle 4 \rangle 7$ DEF $LL1RestrictedCorruption$

For the ELSE case, we assume that an extension is not in progress and show that this refines to an $LL1Restart$ action.

$\langle 3 \rangle 5.$ ASSUME $\neg LL2NVRAM.extensionInProgress$
PROVE $LL1Restart$

The extentials and the first two conjuncts in the $LL1Restart$ action are satisfied by a proof above.

$\langle 4 \rangle 1.$ $\exists\ ll1UntrustedStorage \in LL1UntrustedStorageType,$
$ll1RandomSymmetricKey \in SymmetricKeyType \setminus \{LL1NVRAM.symmetricKey\},$
$ll1Hash \in HashType :$
$\land\ ll1UntrustedStorage.authenticator =$
$GenerateMAC(ll1RandomSymmetricKey, ll1Hash)$
$\land\ LL1RAM' = ll1UntrustedStorage$
BY $\langle 3 \rangle 3$

The disk is unchanged by the $UnchangedDiskLemma$.

$\langle 4 \rangle 2.$ UNCHANGED $LL1Disk$
BY $\langle 2 \rangle 1$, $\langle 3 \rangle 2$, $UnchangedDiskLemma$

The $NVRAM$ is unchanged because the logical history summary is unchanged and the symmetric key is unchanged.

$\langle 4 \rangle 3.$ UNCHANGED $LL1NVRAM$
$\langle 5 \rangle 1.$ UNCHANGED $LL1NVRAM.historySummary$

The logical history summary in the Memoir-Opt $NVRAM$ and $SPCR$ is unchanged.

$\langle 6 \rangle 1.$ UNCHANGED $LL2NVRAMLogicalHistorySummary$

We reveal the definition of the unprimed logical history summary in the Memoir-Opt $NVRAM$ and $SPCR$, given that there is no extension in progress.

$\langle 7 \rangle 1.$ $LL2NVRAMLogicalHistorySummary = [$
$anchor \mapsto LL2NVRAM.historySummaryAnchor,$
$extension \mapsto BaseHashValue]$
$\langle 8 \rangle 1.$ $\neg LL2NVRAM.extensionInProgress$
BY $\langle 3 \rangle 5$
$\langle 8 \rangle 2.$ QED
BY $\langle 8 \rangle 1$ DEF $LL2NVRAMLogicalHistorySummary$

We reveal the definition of the primed logical history summary in the Memoir-Opt $NVRAM$ and $SPCR$, given that there is no extension in progress.

$\langle 7 \rangle 2.$ $LL2NVRAMLogicalHistorySummary' = [$
$anchor \mapsto LL2NVRAM.historySummaryAnchor',$
$extension \mapsto BaseHashValue]$
$\langle 8 \rangle 1.$ $\neg LL2NVRAM.extensionInProgress'$
$\langle 9 \rangle 1.$ $\neg LL2NVRAM.extensionInProgress$
BY $\langle 3 \rangle 5$
$\langle 9 \rangle 2.$ UNCHANGED $LL2NVRAM.extensionInProgress$
BY $\langle 3 \rangle 2$
$\langle 9 \rangle 3.$ QED
BY $\langle 9 \rangle 1$, $\langle 9 \rangle 2$
$\langle 8 \rangle 2.$ QED
BY $\langle 8 \rangle 1$ DEF $LL2NVRAMLogicalHistorySummary$

The history summary anchor in the Memoir-Opt $NVRAM$ is unchanged by a $LL2CorruptSPCR$ action.

$\langle 7 \rangle 3.$ UNCHANGED $LL2NVRAM.historySummaryAnchor$
BY $\langle 3 \rangle 2$

Since the value of $LL2NVRAMLogicalHistorySummary$ is determined entirely by the history summary in the Memoir-Opt $NVRAM$, and since this value is unchanged, the value of $LL2NVRAMLogicalHistorySummary$ is unchanged.

$\langle 7 \rangle 4.$ QED
BY $\langle 7 \rangle 1$, $\langle 7 \rangle 2$, $\langle 7 \rangle 3$
$\langle 6 \rangle 2.$ UNCHANGED $LL2NVRAM.symmetricKey$
BY $\langle 3 \rangle 2$
$\langle 6 \rangle 3.$ UNCHANGED $LL2NVRAM.hashBarrier$
BY $\langle 3 \rangle 2$
$\langle 6 \rangle 4.$ QED
BY $\langle 2 \rangle 1$, $\langle 6 \rangle 1$, $\langle 6 \rangle 2$, $\langle 6 \rangle 3$, $UnchangedNVRAMHistorySummaryLemma$
$\langle 5 \rangle 2.$ UNCHANGED $LL1NVRAM.symmetricKey$
$\langle 6 \rangle 1.$ UNCHANGED $LL2NVRAM.symmetricKey$
BY $\langle 3 \rangle 2$
$\langle 6 \rangle 2.$ QED
BY $\langle 2 \rangle 1$, $\langle 6 \rangle 1$, $UnchangedNVRAMSymmetricKeyLemma$
$\langle 5 \rangle 3.$ $LL1NVRAM \in LL1TrustedStorageType$
BY $\langle 2 \rangle 1$ DEF $LL2Refinement$
$\langle 5 \rangle 4.$ $LL1NVRAM' \in LL1TrustedStorageType$
BY $\langle 2 \rangle 1$ DEF $LL2Refinement$
$\langle 5 \rangle 5.$ QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$, $\langle 5 \rangle 3$, $\langle 5 \rangle 4$, $LL1NVRAMRecordCompositionLemma$

The set of available inputs is unchanged by the $UnchangedAvailableInputsLemma$.

$\langle 4 \rangle 4.$ UNCHANGED $LL1AvailableInputs$
BY $\langle 2 \rangle 1$, $\langle 3 \rangle 2$, $UnchangedAvailableInputsLemma$

The set of observed outputs is unchanged by the $UnchangedObservedOutputsLemma$.

$\langle 4 \rangle 5.$ UNCHANGED $LL1ObservedOutputs$
BY $\langle 2 \rangle 1$, $\langle 3 \rangle 2$, $UnchangedObservedOutputsLemma$

The set of observed authenticators is unchanged by the $UnchangedObservedAuthenticatorsLemma$.

$\langle 4 \rangle 6.$ UNCHANGED $LL1ObservedAuthenticators$
BY $\langle 2 \rangle 1$, $\langle 3 \rangle 2$, $UnchangedObservedAuthenticatorsLemma$

Lastly, we tie together all of the required aspects of the $LL1Restart$ definition.

$\langle 4 \rangle 7.$ QED
BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$, $\langle 4 \rangle 3$, $\langle 4 \rangle 4$, $\langle 4 \rangle 5$, $\langle 4 \rangle 6$ DEF $LL1Restart$

Both THEN and ELSE cases are proven.

$\langle 3 \rangle 6.$ QED
BY $\langle 3 \rangle 4$, $\langle 3 \rangle 5$

A Memoir-Opt $LL2ReadDisk$ action refines to a Memoir-Basic $LL1ReadDisk$ action.

$\langle 2 \rangle 8.$ $LL2ReadDisk \Rightarrow LL1ReadDisk$
$\langle 3 \rangle 1.$ HAVE $LL2ReadDisk$

The primed state of the $LL1RAM$ equals the unprimed state of the $LL1Disk$. The proof is somewhat tedious but completely straightforward.

$\langle 3 \rangle 2.$ $LL1RAM' = LL1Disk$
$\langle 4 \rangle 1.$ $LL2RAM' = LL2Disk$
BY $\langle 3 \rangle 1$ DEF $LL2ReadDisk$
$\langle 4 \rangle 2.$ UNCHANGED $LL2NVRAM.hashBarrier$
BY $\langle 3 \rangle 1$ DEF $LL2ReadDisk$

The primed public state field of the RAM equals the unprimed public state field of the disk.

$\langle 4 \rangle 3.$ $LL1RAM.publicState' = LL1Disk.publicState$
$\langle 5 \rangle 1.$ $LL2RAM.publicState' = LL2Disk.publicState$

BY $\langle 4 \rangle 1$
$\langle 5 \rangle 2$. $LL1Disk.publicState = LL2Disk.publicState$
BY $\langle 2 \rangle 1$ DEF $LL2Refinement$
$\langle 5 \rangle 3$. $LL1RAM.publicState' = LL2RAM.publicState'$
BY $\langle 2 \rangle 1$ DEF $LL2Refinement$
$\langle 5 \rangle 4$. QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$, $\langle 5 \rangle 3$

The primed encrypted private state field of the RAM equals the unprimed encrypted private state field of the disk.

$\langle 4 \rangle 4$. $LL1RAM.privateStateEnc' = LL1Disk.privateStateEnc$
$\langle 5 \rangle 1$. $LL2RAM.privateStateEnc' = LL2Disk.privateStateEnc$
BY $\langle 4 \rangle 1$
$\langle 5 \rangle 2$. $LL1Disk.privateStateEnc = LL2Disk.privateStateEnc$
BY $\langle 2 \rangle 1$ DEF $LL2Refinement$
$\langle 5 \rangle 3$. $LL1RAM.privateStateEnc' = LL2RAM.privateStateEnc'$
BY $\langle 2 \rangle 1$ DEF $LL2Refinement$
$\langle 5 \rangle 4$. QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$, $\langle 5 \rangle 3$

The primed history summary field of the RAM equals the unprimed history summary field of the disk.

$\langle 4 \rangle 5$. $LL1RAM.historySummary' = LL1Disk.historySummary$
$\langle 5 \rangle 1$. $LL2RAM.historySummary' = LL2Disk.historySummary$
BY $\langle 4 \rangle 1$
$\langle 5 \rangle 2$. $HistorySummariesMatch($
$LL1Disk.historySummary,$
$LL2Disk.historySummary,$
$LL2NVRAM.hashBarrier)$
BY $\langle 2 \rangle 1$ DEF $LL2Refinement$
$\langle 5 \rangle 3$. $HistorySummariesMatch($
$LL1RAM.historySummary',$
$LL2RAM.historySummary',$
$LL2NVRAM.hashBarrier')$
BY $\langle 2 \rangle 1$ DEF $LL2Refinement$
$\langle 5 \rangle 4$. QED
$\langle 6 \rangle 1$. $LL1Disk.historySummary \in HashType$
BY $\langle 2 \rangle 1$ DEF $LL2Refinement$, $LL1UntrustedStorageType$
$\langle 6 \rangle 2$. $LL1RAM.historySummary' \in HashType$
BY $\langle 2 \rangle 1$ DEF $LL2Refinement$, $LL1UntrustedStorageType$
$\langle 6 \rangle 3$. $LL2Disk.historySummary \in HistorySummaryType$
BY $\langle 2 \rangle 1$, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$
$\langle 6 \rangle 4$. $LL2NVRAM.hashBarrier \in HashType$
BY $\langle 2 \rangle 1$, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$
$\langle 6 \rangle 5$. QED
BY $\langle 4 \rangle 2$, $\langle 5 \rangle 1$, $\langle 5 \rangle 2$, $\langle 5 \rangle 3$, $\langle 6 \rangle 1$, $\langle 6 \rangle 2$, $\langle 6 \rangle 3$, $\langle 6 \rangle 4$,
$HistorySummariesMatchUniqueLemma$

The primed authenticator field of the RAM equals the unprimed authenticator field of the disk.

$\langle 4 \rangle 6$. $LL1RAM.authenticator' = LL1Disk.authenticator$
$\langle 5 \rangle 1$. $LL2RAM.authenticator' = LL2Disk.authenticator$
BY $\langle 4 \rangle 1$
$\langle 5 \rangle 2$. $\exists\, symmetricKey \in SymmetricKeyType :$
$AuthenticatorsMatch($
$LL1Disk.authenticator,$
$LL2Disk.authenticator,$

$symmetricKey$,
$LL2NVRAM.hashBarrier$)
BY ⟨2⟩1 DEF $LL2Refinement$
⟨5⟩3. $\exists\, symmetricKey \in SymmetricKeyType$:
$AuthenticatorsMatch($
$LL1RAM.authenticator'$,
$LL2RAM.authenticator'$,
$symmetricKey$,
$LL2NVRAM.hashBarrier'$)
BY ⟨2⟩1 DEF $LL2Refinement$
⟨5⟩4. QED
⟨6⟩1. $LL1Disk.authenticator \in MACType$
BY ⟨2⟩1 DEF $LL2Refinement$, $LL1UntrustedStorageType$
⟨6⟩2. $LL1RAM.authenticator' \in MACType$
BY ⟨2⟩1 DEF $LL2Refinement$, $LL1UntrustedStorageType$
⟨6⟩3. $LL2Disk.authenticator \in MACType$
BY ⟨2⟩1, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$
⟨6⟩4. $LL2NVRAM.hashBarrier \in HashType$
BY ⟨2⟩1, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$
⟨6⟩5. QED
BY ⟨4⟩2, ⟨5⟩1, ⟨5⟩2, ⟨5⟩3, ⟨6⟩1, ⟨6⟩2, ⟨6⟩3, ⟨6⟩4,
$AuthenticatorsMatchUniqueLemma$
⟨4⟩7. $LL1RAM' \in LL1UntrustedStorageType$
BY ⟨2⟩1 DEF $LL2Refinement$
⟨4⟩8. $LL1Disk \in LL1UntrustedStorageType$
BY ⟨2⟩1 DEF $LL2Refinement$
⟨4⟩9. QED
BY ⟨4⟩3, ⟨4⟩4, ⟨4⟩5, ⟨4⟩6, ⟨4⟩7, ⟨4⟩8,
$LL1RAMRecordCompositionLemma$, $LL1DiskRecordCompositionLemma$

All variables other than $LL1RAM$ are unchanged, by virtue of the corresponding lemmas.

⟨3⟩3. UNCHANGED $LL1Disk$
BY ⟨2⟩1, ⟨3⟩1, $UnchangedDiskLemma$DEF $LL2ReadDisk$
⟨3⟩4. UNCHANGED $LL1NVRAM$
BY ⟨2⟩1, ⟨3⟩1, $UnchangedNVRAMLemma$DEF $LL2ReadDisk$
⟨3⟩5. UNCHANGED $LL1AvailableInputs$
BY ⟨2⟩1, ⟨3⟩1, $UnchangedAvailableInputsLemma$DEF $LL2ReadDisk$
⟨3⟩6. UNCHANGED $LL1ObservedOutputs$
BY ⟨2⟩1, ⟨3⟩1, $UnchangedObservedOutputsLemma$DEF $LL2ReadDisk$
⟨3⟩7. UNCHANGED $LL1ObservedAuthenticators$
BY ⟨2⟩1, ⟨3⟩1, $UnchangedObservedAuthenticatorsLemma$DEF $LL2ReadDisk$
⟨3⟩8. QED
BY ⟨3⟩2, ⟨3⟩3, ⟨3⟩4, ⟨3⟩5, ⟨3⟩6, ⟨3⟩7 DEF $LL1ReadDisk$

A Memoir-Opt $LL2WriteDisk$ action refines to a Memoir-Basic $LL1WriteDisk$ action.

⟨2⟩9. $LL2WriteDisk \Rightarrow LL1WriteDisk$
⟨3⟩1. HAVE $LL2WriteDisk$

The primed state of the $LL1Disk$ equals the unprimed state of the $LL1RAM$. The proof is somewhat tedious but completely straightforward.

⟨3⟩2. $LL1Disk' = LL1RAM$
⟨4⟩1. $LL2Disk' = LL2RAM$
BY ⟨3⟩1 DEF $LL2WriteDisk$
⟨4⟩2. $\wedge$ UNCHANGED $LL2NVRAM.symmetricKey$
$\wedge$ UNCHANGED $LL2NVRAM.hashBarrier$

BY ⟨3⟩1 DEF $LL2WriteDisk$

The primed public state field of the disk equals the unprimed public state field of the RAM.

⟨4⟩3. $LL1Disk.publicState' = LL1RAM.publicState$
⟨5⟩1. $LL2Disk.publicState' = LL2RAM.publicState$
BY ⟨4⟩1
⟨5⟩2. $LL1RAM.publicState = LL2RAM.publicState$
BY ⟨2⟩1 DEF $LL2Refinement$
⟨5⟩3. $LL1Disk.publicState' = LL2Disk.publicState'$
BY ⟨2⟩1 DEF $LL2Refinement$
⟨5⟩4. QED
BY ⟨5⟩1, ⟨5⟩2, ⟨5⟩3

The primed encrypted private state field of the disk equals the unprimed encrypted private state field of the RAM.

⟨4⟩4. $LL1Disk.privateStateEnc' = LL1RAM.privateStateEnc$
⟨5⟩1. $LL2Disk.privateStateEnc' = LL2RAM.privateStateEnc$
BY ⟨4⟩1
⟨5⟩2. $LL1RAM.privateStateEnc = LL2RAM.privateStateEnc$
BY ⟨2⟩1 DEF $LL2Refinement$
⟨5⟩3. $LL1Disk.privateStateEnc' = LL2Disk.privateStateEnc'$
BY ⟨2⟩1 DEF $LL2Refinement$
⟨5⟩4. QED
BY ⟨5⟩1, ⟨5⟩2, ⟨5⟩3

The primed history summary field of the disk equals the unprimed history summary field of the RAM.

⟨4⟩5. $LL1Disk.historySummary' = LL1RAM.historySummary$
⟨5⟩1. $LL2Disk.historySummary' = LL2RAM.historySummary$
BY ⟨4⟩1
⟨5⟩2. $HistorySummariesMatch($
$LL1RAM.historySummary,$
$LL2RAM.historySummary,$
$LL2NVRAM.hashBarrier)$
BY ⟨2⟩1 DEF $LL2Refinement$
⟨5⟩3. $HistorySummariesMatch($
$LL1Disk.historySummary',$
$LL2Disk.historySummary',$
$LL2NVRAM.hashBarrier')$
BY ⟨2⟩1 DEF $LL2Refinement$
⟨5⟩4. QED
⟨6⟩1. $LL1RAM.historySummary \in HashType$
BY ⟨2⟩1 DEF $LL2Refinement$, $LL1UntrustedStorageType$
⟨6⟩2. $LL1Disk.historySummary' \in HashType$
BY ⟨2⟩1 DEF $LL2Refinement$, $LL1UntrustedStorageType$
⟨6⟩3. $LL2RAM.historySummary \in HistorySummaryType$
BY ⟨2⟩1, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$
⟨6⟩4. $LL2NVRAM.hashBarrier \in HashType$
BY ⟨2⟩1, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$
⟨6⟩5. QED
BY ⟨4⟩2, ⟨5⟩1, ⟨5⟩2, ⟨5⟩3, ⟨6⟩1, ⟨6⟩2, ⟨6⟩3, ⟨6⟩4,
$HistorySummariesMatchUniqueLemma$

The primed authenticator field of the disk equals the unprimed authenticator field of the RAM.

⟨4⟩6. $LL1Disk.authenticator' = LL1RAM.authenticator$
⟨5⟩1. $LL2Disk.authenticator' = LL2RAM.authenticator$
BY ⟨4⟩1

⟨5⟩2. $\exists\, symmetricKey \in SymmetricKeyType :$
$AuthenticatorsMatch($
$LL1RAM.authenticator,$
$LL2RAM.authenticator,$
$symmetricKey,$
$LL2NVRAM.hashBarrier)$
BY ⟨2⟩1 DEF $LL2Refinement$
⟨5⟩3. $\exists\, symmetricKey \in SymmetricKeyType :$
$AuthenticatorsMatch($
$LL1Disk.authenticator',$
$LL2Disk.authenticator',$
$symmetricKey,$
$LL2NVRAM.hashBarrier')$
BY ⟨2⟩1 DEF $LL2Refinement$
⟨5⟩4. QED
⟨6⟩1. $LL1RAM.authenticator \in MACType$
BY ⟨2⟩1 DEF $LL2Refinement$, $LL1UntrustedStorageType$
⟨6⟩2. $LL1Disk.authenticator' \in MACType$
BY ⟨2⟩1 DEF $LL2Refinement$, $LL1UntrustedStorageType$
⟨6⟩3. $LL2RAM.authenticator \in MACType$
BY ⟨2⟩1, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$
⟨6⟩4. $LL2NVRAM.hashBarrier \in HashType$
BY ⟨2⟩1, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$
⟨6⟩5. QED
BY ⟨4⟩2, ⟨5⟩1, ⟨5⟩2, ⟨5⟩3, ⟨6⟩1, ⟨6⟩2, ⟨6⟩3, ⟨6⟩4,
$AuthenticatorsMatchUniqueLemma$
⟨4⟩7. $LL1Disk' \in LL1UntrustedStorageType$
BY ⟨2⟩1 DEF $LL2Refinement$
⟨4⟩8. $LL1RAM \in LL1UntrustedStorageType$
BY ⟨2⟩1 DEF $LL2Refinement$
⟨4⟩9. QED
BY ⟨4⟩3, ⟨4⟩4, ⟨4⟩5, ⟨4⟩6, ⟨4⟩7, ⟨4⟩8,
$LL1DiskRecordCompositionLemma$, $LL1RAMRecordCompositionLemma$

All variables other than $LL1Disk$ are unchanged, by virtue of the corresponding lemmas.

⟨3⟩3. UNCHANGED $LL1RAM$
BY ⟨2⟩1, ⟨3⟩1, $UnchangedRAMLemma$DEF $LL2WriteDisk$
⟨3⟩4. UNCHANGED $LL1NVRAM$
BY ⟨2⟩1, ⟨3⟩1, $UnchangedNVRAMLemma$DEF $LL2WriteDisk$
⟨3⟩5. UNCHANGED $LL1AvailableInputs$
BY ⟨2⟩1, ⟨3⟩1, $UnchangedAvailableInputsLemma$DEF $LL2WriteDisk$
⟨3⟩6. UNCHANGED $LL1ObservedOutputs$
BY ⟨2⟩1, ⟨3⟩1, $UnchangedObservedOutputsLemma$DEF $LL2WriteDisk$
⟨3⟩7. UNCHANGED $LL1ObservedAuthenticators$
BY ⟨2⟩1, ⟨3⟩1, $UnchangedObservedAuthenticatorsLemma$DEF $LL2WriteDisk$
⟨3⟩8. QED
BY ⟨3⟩2, ⟨3⟩3, ⟨3⟩4, ⟨3⟩5, ⟨3⟩6, ⟨3⟩7 DEF $LL1WriteDisk$

A Memoir-Opt $LL2CorruptRAM$ action refines to a Memoir-Basic $LL1CorruptRAM$ action.

⟨2⟩10. $LL2CorruptRAM \Rightarrow LL1CorruptRAM$

We assume the antecedent.

⟨3⟩1. HAVE $LL2CorruptRAM$

We pick a set of variables of the appropriate types that satisfy the $LL2CorruptRAM$ action.

$\langle 3\rangle 2.$ PICK $ll2UntrustedStorage \in LL2UntrustedStorageType,$
$ll2FakeSymmetricKey \in SymmetricKeyType \setminus \{LL2NVRAM.symmetricKey\},$
$ll2Hash \in HashType :$
$LL2CorruptRAM!(ll2UntrustedStorage, ll2FakeSymmetricKey, ll2Hash)$
BY $\langle 3\rangle 1$ DEF $LL2CorruptRAM$

We pick a symmetric key that satisfies the *AuthenticatorsMatch* predicate for the primed states of the authenticators in the RAM variables of the two specs.

$\langle 3\rangle 3.$ PICK $symmetricKey \in SymmetricKeyType :$
$AuthenticatorsMatch($
$LL1RAM.authenticator',$
$LL2RAM.authenticator',$
$symmetricKey,$
$LL2NVRAM.hashBarrier')$
BY $\langle 2\rangle 1$ DEF $LL2Refinement$

We pick a set of variables of the appropriate types that satisfy the quantified *AuthenticatorsMatch* predicate.

$\langle 3\rangle 4.$ PICK $stateHash \in HashType,$
$ll1HistorySummary \in HashType,$
$ll2HistorySummary \in HistorySummaryType :$
$AuthenticatorsMatch($
$LL1RAM.authenticator',$
$LL2RAM.authenticator',$
$symmetricKey,$
$LL2NVRAM.hashBarrier')!($
$stateHash, ll1HistorySummary, ll2HistorySummary)!1$
BY $\langle 3\rangle 3$ DEF $AuthenticatorsMatch$

We re-state the definitions from the LET in *AuthenticatorsMatch*.

$\langle 3\rangle$ $ll1HistoryStateBinding \triangleq Hash(ll1HistorySummary, stateHash)$
$\langle 3\rangle$ $ll2HistorySummaryHash \triangleq Hash(ll2HistorySummary.anchor, ll2HistorySummary.extension)$
$\langle 3\rangle$ $ll2HistoryStateBinding \triangleq Hash(ll2HistorySummaryHash, stateHash)$

We prove the types of the definitions, with help from the *AuthenticatorsMatchDefsTypeSafeLemma*.

$\langle 3\rangle 5.$ $\land\ ll1HistoryStateBinding \in HashType$
$\land\ ll2HistorySummaryHash \in HashType$
$\land\ ll2HistoryStateBinding \in HashType$
BY $\langle 3\rangle 4$, $AuthenticatorsMatchDefsTypeSafeLemma$

We hide the definitions.

$\langle 3\rangle$ HIDE DEF $ll1HistoryStateBinding, ll2HistorySummaryHash, ll2HistoryStateBinding$

We prove each aspect of *LL1CorruptRAM* separately. First, we prove the types of the three existentially quantified variables in the definition of *LL1CorruptRAM*.

$\langle 3\rangle 6.$ $LL1RAM' \in LL1UntrustedStorageType$
BY $\langle 2\rangle 1$ DEF $LL2Refinement$
$\langle 3\rangle 7.$ $ll2FakeSymmetricKey \in SymmetricKeyType \setminus \{LL1NVRAM.symmetricKey\}$
$\langle 4\rangle 1.$ $ll2FakeSymmetricKey \in SymmetricKeyType \setminus \{LL2NVRAM.symmetricKey\}$
BY $\langle 3\rangle 2$
$\langle 4\rangle 2.$ $LL1NVRAM.symmetricKey = LL2NVRAM.symmetricKey$
BY $\langle 2\rangle 1$ DEF $LL2Refinement$
$\langle 4\rangle 3.$ QED
BY $\langle 4\rangle 1$, $\langle 4\rangle 2$
$\langle 3\rangle 8.$ $ll1HistoryStateBinding \in HashType$
BY $\langle 3\rangle 5$

We then prove the disjunction regarding the authenticator field in the primed *LL1RAM* record.

$\langle 3 \rangle 9.$ $\lor$ $LL1RAM.authenticator' \in$
$LL1ObservedAuthenticators$
$\lor$ $LL1RAM.authenticator' =$
$GenerateMAC(ll2FakeSymmetricKey,\ ll1HistoryStateBinding)$

We prove that when the authenticator in the Memoir-Opt spec is in the set of observed authenticators, then the authenticator in the Memoir-Basic spec is in the set of observed authenticators. This follows directly from the *AuthenticatorInSetLemma*, once we prove the preconditions for the lemma.

$\langle 4 \rangle 1.$ ASSUME $ll2UntrustedStorage.authenticator \in$
$LL2ObservedAuthenticators$
PROVE $LL1RAM.authenticator' \in LL1ObservedAuthenticators$

We need to prove some types.

$\langle 5 \rangle 1.$ $LL1RAM.authenticator' \in MACType$
$\langle 6 \rangle 1.$ $LL1RAM' \in LL1UntrustedStorageType$
BY $\langle 2 \rangle 1$ DEF $LL2Refinement$
$\langle 6 \rangle 2.$ QED
BY $\langle 6 \rangle 1$ DEF $LL1UntrustedStorageType$
$\langle 5 \rangle 2.$ $LL2RAM.authenticator' \in MACType$
BY $\langle 2 \rangle 1$, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$
$\langle 5 \rangle 3.$ $LL1ObservedAuthenticators \in$ SUBSET $MACType$
BY $\langle 2 \rangle 1$ DEF $LL2Refinement$
$\langle 5 \rangle 4.$ $LL2ObservedAuthenticators \in$ SUBSET $MACType$
BY $\langle 2 \rangle 1$ DEF $LL2TypeInvariant$
$\langle 5 \rangle 5.$ $LL2NVRAM.symmetricKey \in SymmetricKeyType$
BY $\langle 2 \rangle 1$, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$
$\langle 5 \rangle 6.$ $LL2NVRAM.hashBarrier \in HashType$
BY $\langle 2 \rangle 1$, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$

Then we prove the three conjuncts in the antecedent of the *AuthenticatorInSetLemma*. The first conjunct follows from the refinement.

$\langle 5 \rangle 7.$ $\exists\, symmetricKey1 \in SymmetricKeyType :$
$AuthenticatorsMatch($
$LL1RAM.authenticator',$
$LL2RAM.authenticator',$
$symmetricKey1,$
$LL2NVRAM.hashBarrier)$
$\langle 6 \rangle 1.$ $\exists\, symmetricKey1 \in SymmetricKeyType :$
$AuthenticatorsMatch($
$LL1RAM.authenticator',$
$LL2RAM.authenticator',$
$symmetricKey1,$
$LL2NVRAM.hashBarrier')$
BY $\langle 2 \rangle 1$ DEF $LL2Refinement$
$\langle 6 \rangle 2.$ UNCHANGED $LL2NVRAM.hashBarrier$
BY $\langle 3 \rangle 2$
$\langle 6 \rangle 3.$ QED
BY $\langle 6 \rangle 1$, $\langle 6 \rangle 2$

The second conjunct in the antecedent of the *AuthenticatorInSetLemma* also follows from the refinement.

$\langle 5 \rangle 8.$ $AuthenticatorSetsMatch($
$LL1ObservedAuthenticators,$
$LL2ObservedAuthenticators,$
$LL2NVRAM.symmetricKey,$
$LL2NVRAM.hashBarrier)$

BY $\langle 2 \rangle 1$ DEF *LL2Refinement*

The third conjunct in the antecedent of the *AuthenticatorInSetLemma* follows from the first disjunct in the *LL2CorruptRAM* action.

$\langle 5 \rangle 9$. $LL2RAM.authenticator' \in LL2ObservedAuthenticators$

$\langle 6 \rangle 1$. $LL2RAM.authenticator' = ll2UntrustedStorage.authenticator$

$\langle 7 \rangle 1$. $LL2RAM' = ll2UntrustedStorage$

BY $\langle 3 \rangle 2$

$\langle 7 \rangle 2$. QED

BY $\langle 7 \rangle 1$

$\langle 6 \rangle 2$. $ll2UntrustedStorage.authenticator \in LL2ObservedAuthenticators$

BY $\langle 4 \rangle 1$

$\langle 6 \rangle 3$. QED

BY $\langle 6 \rangle 1$, $\langle 6 \rangle 2$

We then invoke the *AuthenticatorInSetLemma* directly.

$\langle 5 \rangle 10$. QED

BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$, $\langle 5 \rangle 3$, $\langle 5 \rangle 4$, $\langle 5 \rangle 5$, $\langle 5 \rangle 6$, $\langle 5 \rangle 7$, $\langle 5 \rangle 8$, $\langle 5 \rangle 9$, *AuthenticatorInSetLemma*

We prove that when the authenticator in the Memoir-Opt spec is generated as a *MAC* with a fake symmetric key, then the authenticator in the Memoir-Basic spec is generated as a *MAC* with a fake symmetric key. This follows directly from the *AuthenticatorGeneratedLemma*, once we prove the preconditions for the lemma.

$\langle 4 \rangle 2$. ASSUME $ll2UntrustedStorage.authenticator = GenerateMAC(ll2FakeSymmetricKey, ll2Hash)$

PROVE $LL1RAM.authenticator' = GenerateMAC(ll2FakeSymmetricKey, ll1HistoryStateBinding)$

We need to prove some types.

$\langle 5 \rangle 1$. $stateHash \in HashType$

BY $\langle 3 \rangle 4$

$\langle 5 \rangle 2$. $ll1HistorySummary \in HashType$

BY $\langle 3 \rangle 4$

$\langle 5 \rangle 3$. $ll2HistorySummary \in HistorySummaryType$

BY $\langle 3 \rangle 4$

$\langle 5 \rangle 4$. $LL1RAM.authenticator' \in MACType$

$\langle 6 \rangle 1$. $LL1RAM' \in LL1UntrustedStorageType$

BY $\langle 2 \rangle 1$ DEF *LL2Refinement*

$\langle 6 \rangle 2$. QED

BY $\langle 6 \rangle 1$ DEF *LL1UntrustedStorageType*

$\langle 5 \rangle 5$. $LL2RAM.authenticator' \in MACType$

BY $\langle 2 \rangle 1$, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*

$\langle 5 \rangle 6$. $ll2FakeSymmetricKey \in SymmetricKeyType$

$\langle 6 \rangle 1$. $ll2FakeSymmetricKey \in SymmetricKeyType \setminus \{LL2NVRAM.symmetricKey\}$

BY $\langle 3 \rangle 2$

$\langle 6 \rangle 2$. QED

BY $\langle 6 \rangle 1$

$\langle 5 \rangle 7$. $LL2NVRAM.hashBarrier' \in HashType$

BY $\langle 2 \rangle 1$, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*

Then we prove the three conjuncts in the antecedent of the *AuthenticatorGeneratedLemma*. The first conjunct follows from a conjunct in the refinement.

$\langle 5 \rangle 8$. $HistorySummariesMatch(ll1HistorySummary, ll2HistorySummary, LL2NVRAM.hashBarrier')$

BY $\langle 3 \rangle 4$

The second conjunct in the antecedent of the *AuthenticatorGeneratedLemma* mainly follows from the refinement, but we also have to prove that the symmetric key specified by an existential in the refinement matches the fake symmetric key specified in the *LL2CorruptRAM* action. This follows from the *MACUnforgeable* property.

$\langle 5 \rangle 9.$ $AuthenticatorsMatch($
$LL1RAM.authenticator',$
$LL2RAM.authenticator',$
$ll2FakeSymmetricKey,$
$LL2NVRAM.hashBarrier')$

The main precondition for the *MACUnforgeable* property is that the generated *MAC* is validated. We first prove the validation, which follows from the refinement.

$\langle 6 \rangle 1.$ $ValidateMAC(symmetricKey, ll2HistoryStateBinding, LL2RAM.authenticator')$
BY $\langle 3 \rangle 4$ DEF $ll2HistoryStateBinding$, $ll2HistorySummaryHash$

We then prove the generation, which follows from the *LL2CorruptRAM* action.

$\langle 6 \rangle 2.$ $LL2RAM.authenticator' = GenerateMAC(ll2FakeSymmetricKey, ll2Hash)$
$\langle 7 \rangle 1.$ $LL2RAM' = ll2UntrustedStorage$
BY $\langle 3 \rangle 2$
$\langle 7 \rangle 2.$ $ll2UntrustedStorage.authenticator = GenerateMAC(ll2FakeSymmetricKey, ll2Hash)$
BY $\langle 3 \rangle 2$
$\langle 7 \rangle 3.$ QED
BY $\langle 7 \rangle 1$, $\langle 7 \rangle 2$

The remaining preconditions are types.

$\langle 6 \rangle 3.$ $symmetricKey \in SymmetricKeyType$
BY $\langle 3 \rangle 3$
$\langle 6 \rangle 4.$ $ll2FakeSymmetricKey \in SymmetricKeyType$
$\langle 7 \rangle 1.$ $ll2FakeSymmetricKey \in SymmetricKeyType \setminus \{LL2NVRAM.symmetricKey\}$
BY $\langle 3 \rangle 2$
$\langle 7 \rangle 2.$ QED
BY $\langle 7 \rangle 1$
$\langle 6 \rangle 5.$ $ll2HistoryStateBinding \in HashType$
BY $\langle 3 \rangle 5$
$\langle 6 \rangle 6.$ $ll2Hash \in HashType$
BY $\langle 3 \rangle 2$

The *MACUnforgeable* property tells us that the two keys are equal.

$\langle 6 \rangle 7.$ $symmetricKey = ll2FakeSymmetricKey$
BY $\langle 6 \rangle 1$, $\langle 6 \rangle 2$, $\langle 6 \rangle 3$, $\langle 6 \rangle 4$, $\langle 6 \rangle 5$, $\langle 6 \rangle 6$, *MACUnforgeable*
$\langle 6 \rangle 8.$ QED
BY $\langle 3 \rangle 3$, $\langle 6 \rangle 7$

The third conjunct in the antecedent of the *AuthenticatorGeneratedLemma* mainly follows from the *LL2CorruptRAM* action, but we also have to prove that the history state binding specified in the refinement matches the arbitrary hash specified in the *LL2CorruptRAM* action.

$\langle 5 \rangle 10.$ $LL2RAM.authenticator' =$
$GenerateMAC(ll2FakeSymmetricKey, ll2HistoryStateBinding)$

The state authenticaton in the primed Memoir-Opt RAM equals the authenticator specified by the existential *ll2UntrustedStorage* in the *LL2CorruptRAM* action.

$\langle 6 \rangle 1.$ $LL2RAM.authenticator' = ll2UntrustedStorage.authenticator$
$\langle 7 \rangle 1.$ $LL2RAM' = ll2UntrustedStorage$
BY $\langle 3 \rangle 2$
$\langle 7 \rangle 2.$ QED
BY $\langle 7 \rangle 1$

The authenticator specified by the existential *ll2UntrustedStorage* in the *LL2CorruptRAM* action is generated as a *MAC* of the history state binding from the *AuthenticatorsMatch* predicate invoked by the refinement. This follows from the *MACCollisionResistant* property.

$\langle 6 \rangle 2.$ $ll2UntrustedStorage.authenticator =$
$GenerateMAC(ll2FakeSymmetricKey, ll2HistoryStateBinding)$

The main precondition for the *MACUnforgeable* property is that the generated *MAC* is validated. We first prove the validation, which follows from the refinement.

⟨7⟩1. $ValidateMAC(symmetricKey, ll2HistoryStateBinding, LL2RAM.authenticator')$
BY ⟨3⟩4 DEF *ll2HistoryStateBinding*, *ll2HistorySummaryHash*

We then prove the generation, which follows from the *LL2CorruptRAM* action.

⟨7⟩2. $LL2RAM.authenticator' = GenerateMAC(ll2FakeSymmetricKey, ll2Hash)$
⟨8⟩1. $LL2RAM' = ll2UntrustedStorage$
BY ⟨3⟩2
⟨8⟩2. $ll2UntrustedStorage.authenticator = GenerateMAC(ll2FakeSymmetricKey, ll2Hash)$
BY ⟨3⟩2
⟨8⟩3. QED
BY ⟨8⟩1, ⟨8⟩2

The remaining preconditions are types.

⟨7⟩3. $symmetricKey \in SymmetricKeyType$
BY ⟨3⟩3
⟨7⟩4. $ll2FakeSymmetricKey \in SymmetricKeyType$
⟨8⟩1. $ll2FakeSymmetricKey \in SymmetricKeyType \setminus \{LL2NVRAM.symmetricKey\}$
BY ⟨3⟩2
⟨8⟩2. QED
BY ⟨8⟩1
⟨7⟩5. $ll2HistoryStateBinding \in HashType$
BY ⟨3⟩5
⟨7⟩6. $ll2Hash \in HashType$
BY ⟨3⟩2

The *MACCollisionResistant* property tells us that the two hash values are equal.

⟨7⟩7. $ll2Hash = ll2HistoryStateBinding$
BY ⟨7⟩1, ⟨7⟩2, ⟨7⟩3, ⟨7⟩4, ⟨7⟩5, ⟨7⟩6, *MACCollisionResistant*
⟨7⟩8. QED
BY ⟨4⟩2, ⟨7⟩7
⟨6⟩3. QED
BY ⟨6⟩1, ⟨6⟩2

We then invoke the *AuthenticatorGeneratedLemma* directly.

⟨5⟩11. QED
BY ⟨5⟩1, ⟨5⟩2, ⟨5⟩3, ⟨5⟩4, ⟨5⟩5, ⟨5⟩6, ⟨5⟩7, ⟨5⟩8, ⟨5⟩9, ⟨5⟩10,
AuthenticatorGeneratedLemma
DEF *ll1HistoryStateBinding*, *ll2HistorySummaryHash*, *ll2HistoryStateBinding*

From the definition of *LL2CorruptRAM*, the above two cases are exhaustive.

⟨4⟩3. $\lor\ ll2UntrustedStorage.authenticator \in LL2ObservedAuthenticators$
$\lor\ ll2UntrustedStorage.authenticator = GenerateMAC(ll2FakeSymmetricKey, ll2Hash)$
BY ⟨3⟩2
⟨4⟩4. QED
BY ⟨4⟩1, ⟨4⟩2, ⟨4⟩3

All variables other than *LL1RAM* are unchanged, by virtue of the corresponding lemmas.

⟨3⟩10. UNCHANGED *LL1Disk*
BY ⟨2⟩1, ⟨3⟩2, *UnchangedDiskLemma*DEF *LL2CorruptRAM*
⟨3⟩11. UNCHANGED *LL1NVRAM*
BY ⟨2⟩1, ⟨3⟩2, *UnchangedNVRAMLemma*DEF *LL2CorruptRAM*
⟨3⟩12. UNCHANGED *LL1AvailableInputs*

BY ⟨2⟩1, ⟨3⟩2, *UnchangedAvailableInputsLemma* DEF *LL2CorruptRAM*

⟨3⟩13. UNCHANGED *LL1ObservedOutputs*

BY ⟨2⟩1, ⟨3⟩2, *UnchangedObservedOutputsLemma* DEF *LL2CorruptRAM*

⟨3⟩14. UNCHANGED *LL1ObservedAuthenticators*

BY ⟨2⟩1, ⟨3⟩2, *UnchangedObservedAuthenticatorsLemma* DEF *LL2CorruptRAM*

Lastly, we tie together all of the required aspects of the *LL1CorruptRAM* definition.

⟨3⟩15. QED

BY ⟨3⟩6, ⟨3⟩7, ⟨3⟩8, ⟨3⟩9, ⟨3⟩10, ⟨3⟩11, ⟨3⟩12, ⟨3⟩13, ⟨3⟩14 DEF *LL1CorruptRAM*

A Memoir-Opt *LL2CorruptSPCR* action refines to one of two actions in the Memoir-Basic spec. If an extension is in progress at the time the *LL2Restart* occurs, the corruption of the *SPCR* value caused by the *LL2CorruptSPCR* is fatal, so this refines to a *LL1RestrictedCorruption* action in the Memoir-Basic spec, which in turn refines to an *HLDie* action in the high-level spec. On the other hand, if an extension is not in progress at the time the *LL2Restart* occurs, the action refines to a stuttering step in the Memoir-Basic spec.

⟨2⟩11. *LL2CorruptSPCR* $\Rightarrow$

IF *LL2NVRAM.extensionInProgress*

THEN

LL1RestrictedCorruption

ELSE

UNCHANGED *LL1Vars*

We assume the antecedent.

⟨3⟩1. HAVE *LL2CorruptSPCR*

We pick a fake hash that satisfies the *LL2CorruptSPCR* action.

⟨3⟩2. PICK *fakeHash* $\in$ *HashDomain* : *LL2CorruptSPCR*!(*fakeHash*)

BY ⟨3⟩1 DEF *LL2CorruptSPCR*

We re-state the definition from *LL2CorruptSPCR*.

⟨3⟩ *newHistorySummaryExtension* $\triangleq$ *Hash*(*LL2SPCR*, *fakeHash*)

We then hide the definition.

⟨3⟩ HIDE DEF *newHistorySummaryExtension*

Since the refinement is an IF - THEN - ELSE , we prove the THEN and ELSE cases separately. For the THEN case, we assume that an extension is in progress and show that this refines to a *LL1RestrictedCorruption* action.

⟨3⟩3. ASSUME *LL2NVRAM.extensionInProgress*

PROVE *LL1RestrictedCorruption*

One fact that will be useful in several places below is that the extension field in the logical history summary of the Memoir-Opt *NVRAM* and *SPCR* is not equal to the base hash value. We'll separately prove the two cases of whether or not the primed *SPCR* equals the base hash value.

⟨4⟩1. *LL2NVRAMLogicalHistorySummary.extension*′ $\neq$ *BaseHashValue*

One fact that will be useful for both cases is that an extension is in progress in the primed state.

⟨5⟩1. *LL2NVRAM.extensionInProgress*′

⟨6⟩1. *LL2NVRAM.extensionInProgress*

BY ⟨3⟩3

⟨6⟩2. UNCHANGED *LL2NVRAM.extensionInProgress*

BY ⟨3⟩2

⟨6⟩3. QED

BY ⟨6⟩1, ⟨6⟩2

The first case is fairly simple. When an extension is in progress but the *SPCR* equals the base hash value, the logical history summary equals a crazy hash value, because this sitution should never arise during normal operation.

⟨5⟩2. CASE *LL2SPCR*′ = *BaseHashValue*

⟨6⟩1. *LL2NVRAMLogicalHistorySummary.extension*′ = *CrazyHashValue*

⟨7⟩1. *LL2NVRAMLogicalHistorySummary*′ = [

$anchor \mapsto LL2NVRAM.historySummaryAnchor'$,
$extension \mapsto CrazyHashValue]$

BY ⟨5⟩1, ⟨5⟩2 DEF *LL2NVRAMLogicalHistorySummary*

⟨7⟩2. QED

BY ⟨7⟩1

⟨6⟩2. $CrazyHashValue \neq BaseHashValue$

BY *CrazyHashValueUnique*

⟨6⟩3. QED

BY ⟨6⟩1, ⟨6⟩2

The second case is slightly more involved. We will prove this in two steps.

⟨5⟩3. CASE $LL2SPCR' \neq BaseHashValue$

First, we prove that the primed value of the extension field in the Memoir-Opt logical history summary equals the new history summary extension defined in the *LL2CorruptSPCR* action. This follows fairly directly from the definitions of the *LL2CorruptSPCR* action and the *LL2NVRAMLogicalHistorySummary* operator.

⟨6⟩1. $LL2NVRAMLogicalHistorySummary.extension' = newHistorySummaryExtension$

⟨7⟩1. $LL2SPCR' = newHistorySummaryExtension$

BY ⟨3⟩2 DEF *newHistorySummaryExtension*

⟨7⟩2. $LL2NVRAMLogicalHistorySummary' = [$
$anchor \mapsto LL2NVRAM.historySummaryAnchor'$,
$extension \mapsto LL2SPCR']$

BY ⟨5⟩1, ⟨5⟩3 DEF *LL2NVRAMLogicalHistorySummary*

⟨7⟩3. QED

BY ⟨7⟩1, ⟨7⟩2

Second, we prove that the new history summary extension is not equal to the base hash value. This follows because the new history summary extension is generated as a hash by the *LL2CorruptSPCR* action, and the *BaseHashValueUnique* property tells us that no hash value generated by the *Hash* function can equal the base hash value.

⟨6⟩2. $newHistorySummaryExtension \neq BaseHashValue$

⟨7⟩1. $LL2SPCR \in HashDomain$

⟨8⟩1. $LL2SPCR \in HashType$

BY ⟨2⟩1 DEF *LL2TypeInvariant*

⟨8⟩2. QED

BY ⟨8⟩1 DEF *HashDomain*

⟨7⟩2. $fakeHash \in HashDomain$

BY ⟨3⟩2

⟨7⟩3. QED

BY ⟨7⟩1, ⟨7⟩2, *BaseHashValueUnique*DEF *newHistorySummaryExtension*

⟨6⟩3. QED

BY ⟨6⟩1, ⟨6⟩2

The two cases are exhaustive.

⟨5⟩4. QED

BY ⟨5⟩2, ⟨5⟩3

We prove each conjunct of *LL1RestrictedCorruption* separately. First, we prove the conjunct relating to the *NVRAM*.

⟨4⟩2. $LL1RestrictedCorruption!nvram$

The primed value of the history summary field in the Memoir-Basic *NVRAM* serves as our witness for the garbage history summary.

⟨5⟩1. $LL1NVRAM.historySummary' \in HashType$

BY ⟨2⟩1 DEF *LL2Refinement*, *LL1TrustedStorageType*

We prove that the constraint labeled *current* in the *LL1RestrictedCorruption* action is satisfied.

⟨5⟩2. $LL1RestrictedCorruption!nvram!current(LL1NVRAM.historySummary')$

To prove the universally quantified expression, we take a set of variables of the appropriate types.

$\langle 6 \rangle 1.$ TAKE $stateHash1 \in HashType$,
$ll1Authenticator \in LL1ObservedAuthenticators$

We re-state the definition from within the *LL1RestrictedCorruption*!*nvram*!*current* clause.

$\langle 6 \rangle$ $ll1GarbageHistoryStateBinding \triangleq Hash(LL1NVRAM.historySummary', stateHash1)$

We hide the definition.

$\langle 6 \rangle$ HIDE DEF $ll1GarbageHistoryStateBinding$

We need to prove the *nvram*:: *current* conjunct, which asserts that the authenticator is not a valid *MAC* for the history state binding formed from the history summary in the *NVRAM* and any state hash.

$\langle 6 \rangle 2.$ $\neg ValidateMAC($
$LL1NVRAM.symmetricKey$,
$ll1GarbageHistoryStateBinding$,
$ll1Authenticator)$

We will use proof by contradiction.

$\langle 7 \rangle 1.$ SUFFICES
ASSUME
$ValidateMAC($
$LL1NVRAM.symmetricKey$,
$ll1GarbageHistoryStateBinding$,
$ll1Authenticator)$
PROVE
FALSE
OBVIOUS

We first pick, from the set of Memoir-Opt observed authenticators, a Memoir-Opt authenticator that matches the Memoir-Basic authenticator. We know that such a authenticator exists, because the refinement asserts that the sets of observed authenticators match across the two specs.

$\langle 7 \rangle 2.$ PICK $ll2Authenticator \in LL2ObservedAuthenticators$:
$AuthenticatorsMatch($
$ll1Authenticator$,
$ll2Authenticator$,
$LL2NVRAM.symmetricKey$,
$LL2NVRAM.hashBarrier)$

$\langle 8 \rangle 1.$ $ll1Authenticator \in LL1ObservedAuthenticators$
BY $\langle 6 \rangle 1$

$\langle 8 \rangle 2.$ $AuthenticatorSetsMatch($
$LL1ObservedAuthenticators$,
$LL2ObservedAuthenticators$,
$LL2NVRAM.symmetricKey$,
$LL2NVRAM.hashBarrier)$
BY $\langle 2 \rangle 1$ DEF $LL2Refinement$

$\langle 8 \rangle 3.$ QED
BY $\langle 8 \rangle 1$, $\langle 8 \rangle 2$ DEF $AuthenticatorSetsMatch$

We pick a set of variables of the appropriate types that satisfy the quantified *AuthenticatorsMatch* predicate.

$\langle 7 \rangle 3.$ PICK $stateHash2 \in HashType$,
$ll1HistorySummary \in HashType$,
$ll2HistorySummary \in HistorySummaryType$:
$AuthenticatorsMatch($
$ll1Authenticator$,
$ll2Authenticator$,
$LL2NVRAM.symmetricKey$,
$LL2NVRAM.hashBarrier)!($

$stateHash2,\ ll1HistorySummary,\ ll2HistorySummary)!1$
BY $\langle 7\rangle 2$ DEF *AuthenticatorsMatch*

We re-state the definitions from the LET in *AuthenticatorsMatch*.

$\langle 7\rangle$ $ll1HistoryStateBinding \triangleq Hash(ll1HistorySummary,\ stateHash2)$
$\langle 7\rangle$ $ll2HistorySummaryHash \triangleq Hash(ll2HistorySummary.anchor,\ ll2HistorySummary.extension)$
$\langle 7\rangle$ $ll2HistoryStateBinding \triangleq Hash(ll2HistorySummaryHash,\ stateHash2)$

We prove the types of the definitions, with help from the *AuthenticatorsMatchDefsTypeSafeLemma*.

$\langle 7\rangle 4.$ $\land\ ll1HistoryStateBinding \in HashType$
$\land\ ll2HistorySummaryHash \in HashType$
$\land\ ll2HistoryStateBinding \in HashType$
BY $\langle 7\rangle 3$, *AuthenticatorsMatchDefsTypeSafeLemma*

We hide the definitions.

$\langle 7\rangle$ HIDE DEF *ll1HistoryStateBinding*, *ll2HistorySummaryHash*, *ll2HistoryStateBinding*

We prove that the Memoir-Basic s history summary picked to satisfy the *AuthenticatorsMatch* predicate equals the history summary in the primed state of the Memoir-Basic *NVRAM*.

$\langle 7\rangle 5.$ $ll1HistorySummary = LL1NVRAM.historySummary'$

The first step is to show the equality of the history state bindings that bind each of these history summaries to their respective state hashes.

$\langle 8\rangle 1.$ $ll1HistoryStateBinding = ll1GarbageHistoryStateBinding$

By hypothesis, the authenticator is a valid *MAC* for the garbage history state binding.

$\langle 9\rangle 1.$ $ValidateMAC($
$LL2NVRAM.symmetricKey,\ ll1GarbageHistoryStateBinding,\ ll1Authenticator)$
$\langle 10\rangle 1.$ $LL1NVRAM.symmetricKey = LL2NVRAM.symmetricKey$
BY $\langle 2\rangle 1$ DEF *LL2Refinement*
$\langle 10\rangle 2.$ QED
BY $\langle 7\rangle 1$, $\langle 10\rangle 1$

The definition of the *AuthenticatorsMatch* predicate tells us that the Memoir-Basic authenticator was generated as a *MAC* from the history state binding.

$\langle 9\rangle 2.$ $ll1Authenticator = GenerateMAC(LL2NVRAM.symmetricKey,\ ll1HistoryStateBinding)$
BY $\langle 7\rangle 3$ DEF *ll1HistoryStateBinding*

The remaining preconditions are types.

$\langle 9\rangle 3.$ $LL2NVRAM.symmetricKey \in SymmetricKeyType$
BY $\langle 2\rangle 1$, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*
$\langle 9\rangle 4.$ $ll1HistoryStateBinding \in HashType$
BY $\langle 7\rangle 4$
$\langle 9\rangle 5.$ $ll1GarbageHistoryStateBinding \in HashType$
$\langle 10\rangle 1.$ $LL1NVRAM.historySummary' \in HashDomain$
BY $\langle 5\rangle 1$ DEF *HashDomain*
$\langle 10\rangle 2.$ $stateHash1 \in HashDomain$
BY $\langle 5\rangle 1$ DEF *HashDomain*
$\langle 10\rangle 3.$ QED
BY $\langle 10\rangle 1$, $\langle 10\rangle 2$, *HashTypeSafe*DEF *ll1GarbageHistoryStateBinding*

The *MACCollisionResistant* property tells us that the two history state bindings are equal.

$\langle 9\rangle 6.$ QED
BY $\langle 9\rangle 1$, $\langle 9\rangle 2$, $\langle 9\rangle 3$, $\langle 9\rangle 4$, $\langle 9\rangle 5$, *MACCollisionResistant*

By the collision resistance of the hash function, the equality of the history state bindings implies the equality of the history summaries.

$\langle 8\rangle 2.$ QED
$\langle 9\rangle 1.$ $LL1NVRAM.historySummary' \in HashDomain$
$\langle 10\rangle 1.$ $LL1NVRAM.historySummary' \in HashType$

BY $\langle 2\rangle 1$ DEF *LL2Refinement*, *LL1TrustedStorageType*

$\langle 10\rangle 2.$ QED

BY $\langle 10\rangle 1$ DEF *HashDomain*

$\langle 9\rangle 2.$ $ll1HistorySummary \in HashDomain$

$\langle 10\rangle 1.$ $ll1HistorySummary \in HashType$

BY $\langle 7\rangle 3$

$\langle 10\rangle 2.$ QED

BY $\langle 10\rangle 1$ DEF *HashDomain*

$\langle 9\rangle 3.$ $stateHash1 \in HashDomain$

$\langle 10\rangle 1.$ $stateHash1 \in HashType$

BY $\langle 6\rangle 1$

$\langle 10\rangle 2.$ QED

BY $\langle 10\rangle 1$ DEF *HashDomain*

$\langle 9\rangle 4.$ $stateHash2 \in HashDomain$

$\langle 10\rangle 1.$ $stateHash2 \in HashType$

BY $\langle 7\rangle 3$

$\langle 10\rangle 2.$ QED

BY $\langle 10\rangle 1$ DEF *HashDomain*

$\langle 9\rangle 5.$ QED

BY $\langle 8\rangle 1$, $\langle 9\rangle 2$, $\langle 9\rangle 1$, $\langle 9\rangle 4$, $\langle 9\rangle 3$, *HashCollisionResistant*

DEF *ll1HistoryStateBinding*, *ll1GarbageHistoryStateBinding*

The Memoir-Basic s history summary picked to satisfy the *AuthenticatorsMatch* predicate matches the primed logical history summary in the Memoir-Opt *NVRAM* and *SPCR*, by the refinement and the above equality.

$\langle 7\rangle 6.$ $HistorySummariesMatch($
$ll1HistorySummary,$
$LL2NVRAMLogicalHistorySummary',$
$LL2NVRAM.hashBarrier')$

$\langle 8\rangle 1.$ $HistorySummariesMatch($
$LL1NVRAM.historySummary',$
$LL2NVRAMLogicalHistorySummary',$
$LL2NVRAM.hashBarrier')$

BY $\langle 2\rangle 1$ DEF *LL2Refinement*

$\langle 8\rangle 2.$ QED

BY $\langle 7\rangle 5$, $\langle 8\rangle 1$

We prove that the *HistorySummariesMatch* predicate equals the *HistorySummariesMatchRecursion* predicate in this case. We assert each condition required by the definition of the predicate.

$\langle 7\rangle 7.$ $HistorySummariesMatch($
$ll1HistorySummary, LL2NVRAMLogicalHistorySummary', LL2NVRAM.hashBarrier') =$
$HistorySummariesMatchRecursion($
$ll1HistorySummary, LL2NVRAMLogicalHistorySummary', LL2NVRAM.hashBarrier')$

We prove some types, to satisfy the universal quantifiers in *HistorySummariesMatchDefinition*.

$\langle 8\rangle 1.$ $ll1HistorySummary \in HashType$

BY $\langle 7\rangle 3$

$\langle 8\rangle 2.$ $LL2NVRAMLogicalHistorySummary' \in HistorySummaryType$

BY $\langle 2\rangle 1$, *LL2NVRAMLogicalHistorySummaryTypeSafe*

$\langle 8\rangle 3.$ $LL2NVRAM.hashBarrier' \in HashType$

BY $\langle 2\rangle 1$, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*

We prove that the Memoir-Opt logical history summary does not equal the initial history summary.

$\langle 8\rangle$ $ll2InitialHistorySummary \triangleq [anchor \mapsto BaseHashValue, extension \mapsto BaseHashValue]$

$\langle 8\rangle 4.$ $LL2NVRAMLogicalHistorySummary' \neq ll2InitialHistorySummary$

⟨9⟩1. $LL2NVRAMLogicalHistorySummary.extension' \neq BaseHashValue$
BY ⟨4⟩1
⟨9⟩2. $ll2InitialHistorySummary.extension = BaseHashValue$
BY DEF $ll2InitialHistorySummary$
⟨9⟩3. QED
BY ⟨9⟩1, ⟨9⟩2

Finally, from *HistorySummariesMatchDefinition*, we can conclude that the *HistorySummariesMatch* predicate equals the quantified *HistorySummariesMatchRecursion* predicate.

⟨8⟩5. QED
BY ⟨8⟩1, ⟨8⟩2, ⟨8⟩3, ⟨8⟩4, *HistorySummariesMatchDefinition*
DEF $ll2InitialHistorySummary$

We pick values for the existential variables inside the *HistorySummariesMatchRecursion* predicate that satisfy the predicate. We know such variables exist, because the predicate is satisfied by the two previous steps.

⟨7⟩8. PICK $prevInput \in InputType$,
$previousLL1HistorySummary \in HashType$,
$previousLL2HistorySummary \in HistorySummaryType$:
$HistorySummariesMatchRecursion($
$ll1HistorySummary$,
$LL2NVRAMLogicalHistorySummary'$,
$LL2NVRAM.hashBarrier')!($
$prevInput$,
$previousLL1HistorySummary$,
$previousLL2HistorySummary)$

We prove some types, to satisfy the universal quantifiers in *HistorySummariesMatchRecursion* predicate.

⟨8⟩1. $ll1HistorySummary \in HashType$
BY ⟨7⟩3
⟨8⟩2. $LL2NVRAMLogicalHistorySummary' \in HistorySummaryType$
BY ⟨2⟩1, *LL2NVRAMLogicalHistorySummaryTypeSafe*
⟨8⟩3. $LL2NVRAM.hashBarrier' \in HashType$
BY ⟨2⟩1, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*
⟨8⟩4. $HistorySummariesMatchRecursion($
$ll1HistorySummary, LL2NVRAMLogicalHistorySummary', LL2NVRAM.hashBarrier')$
BY ⟨7⟩6, ⟨7⟩7
⟨8⟩5. QED
BY ⟨8⟩1, ⟨8⟩2, ⟨8⟩4 DEF *HistorySummariesMatchRecursion*

One of the conjuncts in the definition of *HistorySummariesMatchRecursion* is that the Memoir-Opt history summary is a successor of a previous history summary.

⟨7⟩9. $LL2HistorySummaryIsSuccessor($
$LL2NVRAMLogicalHistorySummary'$,
$previousLL2HistorySummary$,
$prevInput$,
$LL2NVRAM.hashBarrier')$
BY ⟨7⟩8 DEF *HistorySummariesMatchRecursion*

We re-state the definitions from the LET in *LL2HistorySummaryIsSuccessor*.

⟨7⟩ $successorHistorySummary \triangleq$
$Successor(previousLL2HistorySummary, prevInput, LL2NVRAM.hashBarrier')$
⟨7⟩ $checkpointedSuccessorHistorySummary \triangleq Checkpoint(successorHistorySummary)$

We hide the definitions.

⟨7⟩ HIDE DEF $successorHistorySummary$, $checkpointedSuccessorHistorySummary$

The definition of *LL2HistorySummaryIsSuccessor* tells us that there are two ways that the logical history summary could be a successor. We will prove that neither of these disjuncts is satisfiable.

$\langle 7 \rangle 10.$ $\lor$ $LL2NVRAMLogicalHistorySummary' = successorHistorySummary$
$\quad\lor$ $LL2NVRAMLogicalHistorySummary' = checkpointedSuccessorHistorySummary$
BY $\langle 7 \rangle 9$
DEF *LL2HistorySummaryIsSuccessor*, *successorHistorySummary*, *checkpointedSuccessorHistorySummary*

First, we prove that the logical history summary cannot be a successor.

$\langle 7 \rangle 11.$ $LL2NVRAMLogicalHistorySummary' \neq successorHistorySummary$

We re-state a definition from the LET in the *Successor* operator.

$\langle 8 \rangle$ $securedInput \triangleq Hash(LL2NVRAM.hashBarrier', prevInput)$

We hide the definition.

$\langle 8 \rangle$ HIDE DEF *securedInput*

There is only one sub-step, which is proving that the extension fields of these two records are unequal. We'll separately prove the two cases of whether or not the primed *SPCR* equals the base hash value.

$\langle 8 \rangle 1.$ $LL2NVRAMLogicalHistorySummary.extension' \neq successorHistorySummary.extension$

One fact that will be useful for both cases is that an extension is in progress in the primed state.

$\langle 9 \rangle 1.$ $LL2NVRAM.extensionInProgress'$
$\langle 10 \rangle 1.$ $LL2NVRAM.extensionInProgress$
BY $\langle 3 \rangle 3$
$\langle 10 \rangle 2.$ UNCHANGED $LL2NVRAM.extensionInProgress$
BY $\langle 3 \rangle 2$
$\langle 10 \rangle 3.$ QED
BY $\langle 10 \rangle 1$, $\langle 10 \rangle 2$

The first case is when the *SPCR* equals the base hash value.

$\langle 9 \rangle 2.$ CASE $LL2SPCR' = BaseHashValue$

If an extension is in progress but the *SPCR* equals the base hash value, the logical history summary equals a crazy hash value, because this sitution should never arise during normal operation.

$\langle 10 \rangle 1.$ $LL2NVRAMLogicalHistorySummary.extension' = CrazyHashValue$
$\langle 11 \rangle 1.$ $LL2NVRAMLogicalHistorySummary' = [$
$\quad anchor \mapsto LL2NVRAM.historySummaryAnchor',$
$\quad extension \mapsto CrazyHashValue]$
BY $\langle 9 \rangle 1$, $\langle 9 \rangle 2$ DEF *LL2NVRAMLogicalHistorySummary*
$\langle 11 \rangle 2.$ QED
BY $\langle 11 \rangle 1$

The extension field of the successor history summary is equal to a hash generated by the hash function.

$\langle 10 \rangle 2.$ $successorHistorySummary.extension =$
$\quad Hash(previousLL2HistorySummary.extension, securedInput)$
BY DEF *successorHistorySummary*, *Successor*, *securedInput*

The arguments to the hash function are both in the hash domain.

$\langle 10 \rangle 3.$ $previousLL2HistorySummary.extension \in HashDomain$
$\langle 11 \rangle 1.$ $previousLL2HistorySummary.extension \in HashType$
$\langle 12 \rangle 1.$ $previousLL2HistorySummary \in HistorySummaryType$
BY $\langle 7 \rangle 8$
$\langle 12 \rangle 2.$ QED
BY $\langle 12 \rangle 1$ DEF *HistorySummaryType*
$\langle 11 \rangle 2.$ QED
BY $\langle 11 \rangle 1$ DEF *HashDomain*
$\langle 10 \rangle 4.$ $securedInput \in HashDomain$
$\langle 11 \rangle 1.$ $securedInput \in HashType$

$\langle 12\rangle 1.$ $LL2NVRAM.hashBarrier' \in HashDomain$
$\langle 13\rangle 1.$ $LL2NVRAM.hashBarrier' \in HashType$
BY $\langle 2\rangle 1$, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$
$\langle 13\rangle 2.$ QED
BY $\langle 13\rangle 1$ DEF $HashDomain$
$\langle 12\rangle 2.$ $prevInput \in HashDomain$
$\langle 13\rangle 1.$ $prevInput \in InputType$
BY $\langle 7\rangle 8$
$\langle 13\rangle 2.$ QED
BY $\langle 13\rangle 1$ DEF $HashDomain$
$\langle 12\rangle 3.$ QED
BY $\langle 12\rangle 1$, $\langle 12\rangle 2$, $HashTypeSafe$DEF $securedInput$
$\langle 11\rangle 2.$ QED
BY $\langle 11\rangle 1$ DEF $HashDomain$

The crazy hash value is not equal to any hash value that can be generated by the hash function when operating on arguments within its domain.

$\langle 10\rangle 5.$ QED
BY $\langle 10\rangle 1$, $\langle 10\rangle 2$, $\langle 10\rangle 3$, $\langle 10\rangle 4$, $CrazyHashValueUnique$

We will prove the second case in two steps.

$\langle 9\rangle 3.$ CASE $LL2SPCR' \neq BaseHashValue$

First, we prove that the extension field of the logical history summary is a hash whose second argument is the *fakeHash* from the *LL2CorruptSPCR* action. This follows from the definitions of *LL2CorruptSPCR* and *LL2NVRAMLogicalHistorySummary*.

$\langle 10\rangle 1.$ $LL2NVRAMLogicalHistorySummary.extension' = Hash(LL2SPCR, fakeHash)$
$\langle 11\rangle 1.$ $LL2NVRAMLogicalHistorySummary.extension' = LL2SPCR'$
$\langle 12\rangle 1.$ $LL2NVRAMLogicalHistorySummary' = [$
$anchor \mapsto LL2NVRAM.historySummaryAnchor'$,
$extension \mapsto LL2SPCR']$
BY $\langle 9\rangle 1$, $\langle 9\rangle 3$ DEF $LL2NVRAMLogicalHistorySummary$
$\langle 12\rangle 2.$ QED
BY $\langle 12\rangle 1$
$\langle 11\rangle 2.$ $LL2SPCR' = Hash(LL2SPCR, fakeHash)$
BY $\langle 3\rangle 2$
$\langle 11\rangle 3.$ QED
BY $\langle 11\rangle 1$, $\langle 11\rangle 2$

Second, we prove that the extension field of the successor history summary from *LL2HistorySummaryIsSuccessor* is a hash whose second argument is the secured input from the *Successor* operator. This follows directly from the definition of *successorHistorySummary*.

$\langle 10\rangle 2.$ $successorHistorySummary.extension =$
$Hash(previousLL2HistorySummary.extension, securedInput)$
BY DEF $successorHistorySummary$, $Successor$, $securedInput$

Third, we prove that the two hashes are unequal. We will use the *HashCollisionResistant* property, along with the fact (which we will prove in a sub-step) that the second arguments to the two hash functions are unequal.

$\langle 10\rangle 3.$ $Hash(LL2SPCR, fakeHash) \neq$
$Hash(previousLL2HistorySummary.extension, securedInput)$

To employ the *HashCollisionResistant* property, we need to prove some types.

$\langle 11\rangle 1.$ $LL2SPCR \in HashDomain$
$\langle 12\rangle 1.$ $LL2SPCR \in HashType$
BY $\langle 2\rangle 1$ DEF $LL2TypeInvariant$
$\langle 12\rangle 2.$ QED

BY $\langle 12\rangle 1$ DEF *HashDomain*

$\langle 11\rangle 2$. $fakeHash \in HashDomain$

BY $\langle 3\rangle 2$

$\langle 11\rangle 3$. $previousLL2HistorySummary.extension \in HashDomain$

$\langle 12\rangle 1$. $previousLL2HistorySummary.extension \in HashType$

$\langle 13\rangle 1$. $previousLL2HistorySummary \in HistorySummaryType$

BY $\langle 7\rangle 8$

$\langle 13\rangle 2$. QED

BY $\langle 13\rangle 1$ DEF *HistorySummaryType*

$\langle 12\rangle 2$. QED

BY $\langle 12\rangle 1$ DEF *HashDomain*

$\langle 11\rangle 4$. $securedInput \in HashDomain$

$\langle 12\rangle 1$. $securedInput \in HashType$

$\langle 13\rangle 1$. $LL2NVRAM.hashBarrier' \in HashDomain$

$\langle 14\rangle 1$. $LL2NVRAM.hashBarrier' \in HashType$

BY $\langle 2\rangle 1$, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*

$\langle 14\rangle 2$. QED

BY $\langle 14\rangle 1$ DEF *HashDomain*

$\langle 13\rangle 2$. $prevInput \in HashDomain$

$\langle 14\rangle 1$. $prevInput \in InputType$

BY $\langle 7\rangle 8$

$\langle 14\rangle 2$. QED

BY $\langle 14\rangle 1$ DEF *HashDomain*

$\langle 13\rangle 3$. QED

BY $\langle 13\rangle 1$, $\langle 13\rangle 2$, *HashTypeSafe*DEF *securedInput*

$\langle 12\rangle 2$. QED

BY $\langle 12\rangle 1$ DEF *HashDomain*

Then we need to prove that the second arguments to the hash functions are unequal. We will employ the restriction on fake hash values from the definition of *LL2CorruptSPCR*.

$\langle 11\rangle 5$. $fakeHash \neq securedInput$

$\langle 12\rangle 1$. $\forall fakeInput \in InputType :$
$fakeHash \neq Hash(LL2NVRAM.hashBarrier', fakeInput)$

$\langle 13\rangle 1$. $\forall fakeInput \in InputType :$
$fakeHash \neq Hash(LL2NVRAM.hashBarrier, fakeInput)$

BY $\langle 3\rangle 2$

$\langle 13\rangle 2$. UNCHANGED $LL2NVRAM.hashBarrier$

BY $\langle 3\rangle 2$

$\langle 13\rangle 3$. QED

BY $\langle 13\rangle 1$, $\langle 13\rangle 2$

$\langle 12\rangle 2$. $prevInput \in InputType$

BY $\langle 7\rangle 8$

$\langle 12\rangle 3$. QED

BY $\langle 12\rangle 1$, $\langle 12\rangle 2$ DEF *securedInput*

$\langle 11\rangle 6$. QED

Ideally, this QED step should just read:

BY $\langle 11\rangle 1$, $\langle 11\rangle 2$, $\langle 11\rangle 3$, $\langle 11\rangle 4$, $\langle 11\rangle 5$, *HashCollisionResistant*

However, the prover seems to get a little confused in this instance. We make life easier for the prover by definining some local variables and hiding their definitions before appealing to the *HashCollisionResistant* assumption.

$\langle 12\rangle\ h1a \triangleq LL2SPCR$

$\langle 12\rangle\ h2a \triangleq fakeHash$

$\langle 12 \rangle$ $h1b \triangleq$ *previousLL2HistorySummary.extension*
$\langle 12 \rangle$ $h2b \triangleq$ *securedInput*
$\langle 12 \rangle 1.$ $h1a \in HashDomain$
BY $\langle 11 \rangle 1$
$\langle 12 \rangle 2.$ $h2a \in HashDomain$
BY $\langle 11 \rangle 2$
$\langle 12 \rangle 3.$ $h1b \in HashDomain$
BY $\langle 11 \rangle 3$
$\langle 12 \rangle 4.$ $h2b \in HashDomain$
BY $\langle 11 \rangle 4$
$\langle 12 \rangle 5.$ $h2a \neq h2b$
BY $\langle 11 \rangle 5$
$\langle 12 \rangle 6.$ $Hash(h1a, h2a) \neq Hash(h1b, h2b)$
$\langle 13 \rangle$ HIDE DEF $h1a, h2a, h1b, h2b$
$\langle 13 \rangle 1.$ QED
BY $\langle 12 \rangle 1$, $\langle 12 \rangle 2$, $\langle 12 \rangle 3$, $\langle 12 \rangle 4$, $\langle 12 \rangle 5$, *HashCollisionResistant*
$\langle 12 \rangle 7.$ QED
BY $\langle 12 \rangle 6$
$\langle 10 \rangle 4.$ QED
BY $\langle 10 \rangle 1$, $\langle 10 \rangle 2$, $\langle 10 \rangle 3$

The two cases are exhaustive.

$\langle 9 \rangle 4.$ QED
BY $\langle 9 \rangle 2$, $\langle 9 \rangle 3$
$\langle 8 \rangle 2.$ QED
BY $\langle 8 \rangle 1$

Second, we prove that the logical history summary cannot be a checkpoint.

$\langle 7 \rangle 12.$ $LL2NVRAMLogicalHistorySummary' \neq checkpointedSuccessorHistorySummary$

The extension field of the logical history summary does not equal the base hash value, as we proved above.

$\langle 8 \rangle 1.$ $LL2NVRAMLogicalHistorySummary.extension' \neq BaseHashValue$
BY $\langle 4 \rangle 1$

The extension field of the checkpointed successor does equal the base hash value. This follows from the *CheckpointHasBaseExtensionLemma*.

$\langle 8 \rangle 2.$ $checkpointedSuccessorHistorySummary.extension = BaseHashValue$
$\langle 9 \rangle 1.$ $successorHistorySummary \in HistorySummaryType$
$\langle 10 \rangle 1.$ $previousLL2HistorySummary \in HistorySummaryType$
BY $\langle 7 \rangle 8$
$\langle 10 \rangle 2.$ $prevInput \in InputType$
BY $\langle 7 \rangle 8$
$\langle 10 \rangle 3.$ $LL2NVRAM.hashBarrier' \in HashType$
BY $\langle 2 \rangle 1$, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*
$\langle 10 \rangle 4.$ QED
BY $\langle 10 \rangle 1$, $\langle 10 \rangle 2$, $\langle 10 \rangle 3$, *SuccessorTypeSafe*DEF *successorHistorySummary*
$\langle 9 \rangle 2.$ QED
BY $\langle 9 \rangle 1$, *CheckpointHasBaseExtensionLemma*
DEF *checkpointedSuccessorHistorySummary*
$\langle 8 \rangle 3.$ QED
BY $\langle 8 \rangle 1$, $\langle 8 \rangle 2$

We thus have a contradiction.

$\langle 7 \rangle 13.$ QED
BY $\langle 7 \rangle 10$, $\langle 7 \rangle 11$, $\langle 7 \rangle 12$

$\langle 6 \rangle 3.$ QED
  BY $\langle 6 \rangle 2$ DEF $ll1GarbageHistoryStateBinding$

We prove that the constraint labeled *previous* in the *LL1RestrictedCorruption* action is satisfied.

$\langle 5 \rangle 3.$ $LL1RestrictedCorruption!nvram!previous(LL1NVRAM.historySummary')$

To prove the universally quantified expression, we take a set of variables of the appropriate types.

$\langle 6 \rangle 1.$ TAKE $stateHash1 \in HashType,$
$ll1Authenticator \in LL1ObservedAuthenticators,$
$ll1SomeHistorySummary \in HashType,$
$someInput \in InputType$

We re-state the definition from within the *LL1RestrictedCorruption*!*nvram*!*previous* clause.

$\langle 6 \rangle$ $ll1SomeHistoryStateBinding \triangleq Hash(ll1SomeHistorySummary, stateHash1)$

We hide the definition.

$\langle 6 \rangle$ HIDE DEF $ll1SomeHistoryStateBinding$

We need to prove the *nvram*:: *previous* conjunct, which asserts an implication. It suffices to assume the antecedent and prove the consequent.

$\langle 6 \rangle 2.$ SUFFICES
ASSUME $LL1NVRAM.historySummary' = Hash(ll1SomeHistorySummary, someInput)$
PROVE $\neg ValidateMAC($
$LL1NVRAM.symmetricKey,$
$ll1SomeHistoryStateBinding,$
$ll1Authenticator)$
  BY DEF $ll1SomeHistoryStateBinding$

The consequent of the *nvram*:: *previous* conjunct asserts that the authenticator is not a valid *MAC* for the history state binding formed from any predecessor of the history summary in the *NVRAM* and any state hash.

$\langle 6 \rangle 3.$ $\neg ValidateMAC($
$LL1NVRAM.symmetricKey,$
$ll1SomeHistoryStateBinding,$
$ll1Authenticator)$

We will use proof by contradiction.

$\langle 7 \rangle 1.$ SUFFICES
ASSUME
$ValidateMAC($
$LL1NVRAM.symmetricKey,$
$ll1SomeHistoryStateBinding,$
$ll1Authenticator)$
PROVE
FALSE
  OBVIOUS

We first pick, from the set of Memoir-Opt observed authenticators, a Memoir-Opt authenticator that matches the Memoir-Basic authenticator. We know that such a authenticator exists, because the refinement asserts that the sets of observed authenticators match across the two specs.

$\langle 7 \rangle 2.$ PICK $ll2Authenticator \in LL2ObservedAuthenticators:$
$AuthenticatorsMatch($
$ll1Authenticator,$
$ll2Authenticator,$
$LL2NVRAM.symmetricKey,$
$LL2NVRAM.hashBarrier)$

$\langle 8 \rangle 1.$ $ll1Authenticator \in LL1ObservedAuthenticators$
  BY $\langle 6 \rangle 1$

$\langle 8 \rangle 2.$ $AuthenticatorSetsMatch($

$LL1ObservedAuthenticators$,
$LL2ObservedAuthenticators$,
$LL2NVRAM.symmetricKey$,
$LL2NVRAM.hashBarrier$)
BY ⟨2⟩1 DEF $LL2Refinement$
⟨8⟩3. QED
BY ⟨8⟩1, ⟨8⟩2 DEF $AuthenticatorSetsMatch$

We pick a set of variables of the appropriate types that satisfy the quantified *AuthenticatorsMatch* predicate.

⟨7⟩3. PICK $stateHash2 \in HashType$,
$ll1HistorySummary \in HashType$,
$ll2HistorySummary \in HistorySummaryType$:
$AuthenticatorsMatch($
$ll1Authenticator$,
$ll2Authenticator$,
$LL2NVRAM.symmetricKey$,
$LL2NVRAM.hashBarrier)!($
$stateHash2$, $ll1HistorySummary$, $ll2HistorySummary)!1$
BY ⟨7⟩2 DEF $AuthenticatorsMatch$

We re-state the definitions from the LET in *AuthenticatorsMatch*.

⟨7⟩ $ll1HistoryStateBinding \triangleq Hash(ll1HistorySummary, stateHash2)$
⟨7⟩ $ll2HistorySummaryHash \triangleq Hash(ll2HistorySummary.anchor, ll2HistorySummary.extension)$
⟨7⟩ $ll2HistoryStateBinding \triangleq Hash(ll2HistorySummaryHash, stateHash2)$

We prove the types of the definitions, with help from the *AuthenticatorsMatchDefsTypeSafeLemma*.

⟨7⟩4. $\wedge$ $ll1HistoryStateBinding \in HashType$
$\wedge$ $ll2HistorySummaryHash \in HashType$
$\wedge$ $ll2HistoryStateBinding \in HashType$
BY ⟨7⟩3, $AuthenticatorsMatchDefsTypeSafeLemma$

We hide the definitions.

⟨7⟩ HIDE DEF $ll1HistoryStateBinding$, $ll2HistorySummaryHash$, $ll2HistoryStateBinding$

We prove that the Memoir-Basic s history summary picked to satisfy the *AuthenticatorsMatch* predicate equals the history summary in the primed state of the Memoir-Basic *NVRAM*.

⟨7⟩5. $ll1HistorySummary = ll1SomeHistorySummary$

The first step is to show the equality of the historyp state bindings that bind each of these history summaries to their respective state hashes.

⟨8⟩1. $ll1HistoryStateBinding = ll1SomeHistoryStateBinding$

By hypothesis, the authenticator is a valid *MAC* for the garbage history state binding.

⟨9⟩1. $ValidateMAC(LL2NVRAM.symmetricKey, ll1SomeHistoryStateBinding, ll1Authenticator)$
⟨10⟩1. $LL1NVRAM.symmetricKey = LL2NVRAM.symmetricKey$
BY ⟨2⟩1 DEF $LL2Refinement$
⟨10⟩2. QED
BY ⟨7⟩1, ⟨10⟩1

The definition of the *AuthenticatorsMatch* predicate tells us that the Memoir-Basic authenticator was generated as a *MAC* from the history state binding.

⟨9⟩2. $ll1Authenticator = GenerateMAC(LL2NVRAM.symmetricKey, ll1HistoryStateBinding)$
BY ⟨7⟩3 DEF $ll1HistoryStateBinding$

The remaining preconditions are types.

⟨9⟩3. $LL2NVRAM.symmetricKey \in SymmetricKeyType$
BY ⟨2⟩1, $LL2SubtypeImplicationLemma$DEF $LL2SubtypeImplication$
⟨9⟩4. $ll1HistoryStateBinding \in HashType$
BY ⟨7⟩4

⟨9⟩5. $ll1SomeHistoryStateBinding \in HashType$

⟨10⟩1. $ll1SomeHistorySummary \in HashDomain$

⟨11⟩1. $ll1SomeHistorySummary \in HashType$

BY ⟨6⟩1

⟨11⟩2. QED

BY ⟨11⟩1 DEF *HashDomain*

⟨10⟩2. $stateHash1 \in HashDomain$

BY ⟨5⟩1 DEF *HashDomain*

⟨10⟩3. QED

BY ⟨10⟩1, ⟨10⟩2, *HashTypeSafe*DEF *ll1SomeHistoryStateBinding*

The *MACCollisionResistant* property tells us that the two history state bindings are equal.

⟨9⟩6. QED

BY ⟨9⟩1, ⟨9⟩2, ⟨9⟩3, ⟨9⟩4, ⟨9⟩5, *MACCollisionResistant*

By the collision resistance of the hash function, the equality of the history state bindings implies the equality of the history summaries.

⟨8⟩2. QED

⟨9⟩1. $ll1SomeHistorySummary \in HashDomain$

⟨10⟩1. $ll1SomeHistorySummary \in HashType$

BY ⟨6⟩1

⟨10⟩2. QED

BY ⟨10⟩1 DEF *HashDomain*

⟨9⟩2. $ll1HistorySummary \in HashDomain$

⟨10⟩1. $ll1HistorySummary \in HashType$

BY ⟨7⟩3

⟨10⟩2. QED

BY ⟨10⟩1 DEF *HashDomain*

⟨9⟩3. $stateHash1 \in HashDomain$

⟨10⟩1. $stateHash1 \in HashType$

BY ⟨6⟩1

⟨10⟩2. QED

BY ⟨10⟩1 DEF *HashDomain*

⟨9⟩4. $stateHash2 \in HashDomain$

⟨10⟩1. $stateHash2 \in HashType$

BY ⟨7⟩3

⟨10⟩2. QED

BY ⟨10⟩1 DEF *HashDomain*

⟨9⟩5. QED

BY ⟨8⟩1, ⟨9⟩2, ⟨9⟩1, ⟨9⟩4, ⟨9⟩3, *HashCollisionResistant*
DEF *ll1HistoryStateBinding*, *ll1SomeHistoryStateBinding*

We pick a value for the Memoir-Opt previous-inputs-summary existential variable inside the *HistorySummariesMatchRecursion* predicate that satisfies this predicate for (1) the Memoir-Basic s history summary picked to satisfy the *AuthenticatorsMatch* predicate and (2) the input taken from the universal quantifier in the *previous* conjunct in the *LL1RestrictedCorruption* action.

⟨7⟩6. PICK $previousLL2HistorySummary \in HistorySummaryType$:

$$
\begin{array}{l}
HistorySummariesMatchRecursion(\\
\quad LL1NVRAM.historySummary', \\
\quad LL2NVRAMLogicalHistorySummary', \\
\quad LL2NVRAM.hashBarrier')!(\\
\quad someInput, \\
\quad ll1SomeHistorySummary, \\
\quad previousLL2HistorySummary)
\end{array}
$$

The Memoir-Basic s history summary picked to satisfy the *AuthenticatorsMatch* predicate matches the primed logical history summary in the Memoir-Opt *NVRAM* and *SPCR*, by the refinement and the above equality.

$\langle 8 \rangle 1.$ *HistorySummariesMatch*(
$LL1NVRAM.historySummary'$,
$LL2NVRAMLogicalHistorySummary'$,
$LL2NVRAM.hashBarrier'$)

$\langle 9 \rangle 1.$ *HistorySummariesMatch*(
$LL1NVRAM.historySummary'$,
$LL2NVRAMLogicalHistorySummary'$,
$LL2NVRAM.hashBarrier'$)
BY $\langle 2 \rangle 1$ DEF *LL2Refinement*

$\langle 9 \rangle 2.$ QED
BY $\langle 9 \rangle 1$

We prove that the *HistorySummariesMatch* predicate equals the *HistorySummariesMatchRecursion* predicate in this case. We assert each condition required by the definition of the predicate.

$\langle 8 \rangle 2.$ *HistorySummariesMatch*(
$LL1NVRAM.historySummary'$,
$LL2NVRAMLogicalHistorySummary'$,
$LL2NVRAM.hashBarrier'$) =
HistorySummariesMatchRecursion(
$LL1NVRAM.historySummary'$,
$LL2NVRAMLogicalHistorySummary'$,
$LL2NVRAM.hashBarrier'$)

We prove some types, to satisfy the universal quantifiers in *HistorySummariesMatchDefinition*.

$\langle 9 \rangle 1.$ $LL1NVRAM.historySummary' \in HashType$
BY $\langle 2 \rangle 1$ DEF *LL2Refinement*, *LL1TrustedStorageType*

$\langle 9 \rangle 2.$ $LL2NVRAMLogicalHistorySummary' \in HistorySummaryType$
BY $\langle 2 \rangle 1$, *LL2NVRAMLogicalHistorySummaryTypeSafe*

$\langle 9 \rangle 3.$ $LL2NVRAM.hashBarrier' \in HashType$
BY $\langle 2 \rangle 1$, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*

We prove that the Memoir-Opt logical history summary does not equal the initial history summary.

$\langle 9 \rangle$ $ll2InitialHistorySummary \triangleq [anchor \mapsto BaseHashValue, extension \mapsto BaseHashValue]$

$\langle 9 \rangle 4.$ $LL2NVRAMLogicalHistorySummary' \neq ll2InitialHistorySummary$

$\langle 10 \rangle 1.$ $LL2NVRAMLogicalHistorySummary.extension' \neq BaseHashValue$
BY $\langle 4 \rangle 1$

$\langle 10 \rangle 2.$ $ll2InitialHistorySummary.extension = BaseHashValue$
BY DEF *ll2InitialHistorySummary*

$\langle 10 \rangle 3.$ QED
BY $\langle 10 \rangle 1$, $\langle 10 \rangle 2$

Finally, from *HistorySummariesMatchDefinition*, we can conclude that the *HistorySummariesMatch* predicate equals the quantified *HistorySummariesMatchRecursion* predicate.

$\langle 9 \rangle 5.$ QED
BY $\langle 9 \rangle 1$, $\langle 9 \rangle 2$, $\langle 9 \rangle 3$, $\langle 9 \rangle 4$, *HistorySummariesMatchDefinition*
DEF *ll2InitialHistorySummary*

We pick values for the remaining two existential variables inside the *HistorySummariesMatchRecursion* predicate that satisfy the predicate. We know such variables exist, because the predicate is satisfied by the two previous steps.

$\langle 8 \rangle 3.$ PICK $prevInput \in InputType$,
$previousLL1HistorySummary \in HashType$,
$previousLL2HistorySummary \in HistorySummaryType$:

$HistorySummariesMatchRecursion($
$LL1NVRAM.historySummary'$,
$LL2NVRAMLogicalHistorySummary'$,
$LL2NVRAM.hashBarrier')!($
$prevInput$,
$previousLL1HistorySummary$,
$previousLL2HistorySummary)$

We prove some types, to satisfy the universal quantifiers in *HistorySummariesMatchRecursion* predicate.

⟨9⟩1. $ll1HistorySummary \in HashType$
BY ⟨7⟩3
⟨9⟩2. $LL2NVRAMLogicalHistorySummary' \in HistorySummaryType$
BY ⟨2⟩1, *LL2NVRAMLogicalHistorySummaryTypeSafe*
⟨9⟩3. $LL2NVRAM.hashBarrier' \in HashType$
BY ⟨2⟩1, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*
⟨9⟩4. $HistorySummariesMatchRecursion($
$LL1NVRAM.historySummary'$,
$LL2NVRAMLogicalHistorySummary'$,
$LL2NVRAM.hashBarrier')$
BY ⟨8⟩1, ⟨8⟩2
⟨9⟩5. QED
BY ⟨9⟩1, ⟨9⟩2, ⟨9⟩4 DEF *HistorySummariesMatchRecursion*

We prove that the existential variables for the previous input and the previous history summary in the above pick are equal to the input and history summary taken from the universal quantifiers in the *previous* conjunct in the *LL1RestrictedCorruption* action. We use the *HashCollisionResistant* property.

⟨8⟩4. $\land\ ll1SomeHistorySummary = previousLL1HistorySummary$
$\land\ someInput = prevInput$

We prove the necessary types for the *HashCollisionResistant* property.

⟨9⟩1. $ll1SomeHistorySummary \in HashDomain$
⟨10⟩1. $ll1SomeHistorySummary \in HashType$
BY ⟨6⟩1
⟨10⟩2. QED
BY ⟨10⟩1 DEF *HashDomain*
⟨9⟩2. $someInput \in HashDomain$
⟨10⟩1. $someInput \in InputType$
BY ⟨6⟩1
⟨10⟩2. QED
BY ⟨10⟩1 DEF *HashDomain*
⟨9⟩3. $previousLL1HistorySummary \in HashDomain$
⟨10⟩1. $previousLL1HistorySummary \in HashType$
BY ⟨8⟩3
⟨10⟩2. QED
BY ⟨10⟩1 DEF *HashDomain*
⟨9⟩4. $prevInput \in HashDomain$
⟨10⟩1. $prevInput \in InputType$
BY ⟨8⟩3
⟨10⟩2. QED
BY ⟨10⟩1 DEF *HashDomain*

The hashes are equal, because each is equal to the history summary in the primed Memoir-Basic *NVRAM*.

⟨9⟩5. $Hash(ll1SomeHistorySummary, someInput) =$
$Hash(previousLL1HistorySummary, prevInput)$

The hash of the taken history summary and input are equal to the history summary in the primed Memoir-Basic *NVRAM* by assumption of the antecedent in the *previous* conjunct in the *LL1RestrictedCorruption* action.

$\langle 10\rangle 1.$ $LL1NVRAM.historySummary' = Hash(ll1SomeHistorySummary, someInput)$
BY $\langle 6\rangle 2$

The hash of the picked history summary and input are equal to the history summary in the primed Memoir-Basic *NVRAM* by the definition of the *HistorySummariesMatchRecursion* predicate.

$\langle 10\rangle 2.$ $LL1NVRAM.historySummary' = Hash(previousLL1HistorySummary, prevInput)$
BY $\langle 8\rangle 3$

$\langle 10\rangle 3.$ QED
BY $\langle 10\rangle 1$, $\langle 10\rangle 2$

$\langle 9\rangle 6.$ QED
BY $\langle 9\rangle 1$, $\langle 9\rangle 2$, $\langle 9\rangle 3$, $\langle 9\rangle 4$, $\langle 9\rangle 5$, *HashCollisionResistant*

$\langle 8\rangle 5.$ QED
BY $\langle 8\rangle 3$, $\langle 8\rangle 4$

One of the conjuncts in the definition of *HistorySummariesMatchRecursion* is that the Memoir-Opt history summary is a successor of a previous history summary.

$\langle 7\rangle 7.$ $LL2HistorySummaryIsSuccessor($
$LL2NVRAMLogicalHistorySummary',$
$previousLL2HistorySummary,$
$someInput,$
$LL2NVRAM.hashBarrier')$
BY $\langle 7\rangle 6$ DEF *HistorySummariesMatchRecursion*

We re-state the definitions from the LET in *LL2HistorySummaryIsSuccessor*.

$\langle 7\rangle$ $successorHistorySummary \triangleq$
$Successor(previousLL2HistorySummary, someInput, LL2NVRAM.hashBarrier')$

$\langle 7\rangle$ $checkpointedSuccessorHistorySummary \triangleq Checkpoint(successorHistorySummary)$

We hide the definitions.

$\langle 7\rangle$ HIDE DEF *successorHistorySummary*, *checkpointedSuccessorHistorySummary*

The definition of *LL2HistorySummaryIsSuccessor* tells us that there are two ways that the logical history summary could be a successor. We will prove that neither of these disjuncts is satisfiable.

$\langle 7\rangle 8.$ $\lor\ LL2NVRAMLogicalHistorySummary' = successorHistorySummary$
$\lor\ LL2NVRAMLogicalHistorySummary' = checkpointedSuccessorHistorySummary$
BY $\langle 7\rangle 7$
DEF *LL2HistorySummaryIsSuccessor*, *successorHistorySummary*,
checkpointedSuccessorHistorySummary

First, we prove that the logical history summary cannot be a successor.

$\langle 7\rangle 9.$ $LL2NVRAMLogicalHistorySummary' \neq successorHistorySummary$

We re-state a definition from the LET in the *Successor* operator.

$\langle 8\rangle$ $securedInput \triangleq Hash(LL2NVRAM.hashBarrier', someInput)$

We hide the definition.

$\langle 8\rangle$ HIDE DEF *securedInput*

There is only one sub-step, which is proving that the extension fields of these two records are unequal.

$\langle 8\rangle 1.$ $LL2NVRAMLogicalHistorySummary.extension' \neq successorHistorySummary.extension$

First, we prove that the extension field of the logical history summary is a hash whose second argument is the *fakeHash* from the *LL2CorruptSPCR* action. This follows from the definitions of *LL2CorruptSPCR* and *LL2NVRAMLogicalHistorySummary*.

$\langle 9\rangle 1.$ $LL2NVRAMLogicalHistorySummary.extension' = Hash(LL2SPCR, fakeHash)$

$\langle 10\rangle 1.$ $LL2SPCR' = Hash(LL2SPCR, fakeHash)$
BY $\langle 3\rangle 2$

$\langle 10\rangle 2.$ $LL2NVRAMLogicalHistorySummary.extension' = LL2SPCR'$

$\langle 11\rangle 1.$ $LL2NVRAMLogicalHistorySummary' = [$
$anchor \mapsto LL2NVRAM.historySummaryAnchor',$
$extension \mapsto LL2SPCR']$

$\langle 12\rangle 1.$ $LL2NVRAM.extensionInProgress'$

$\langle 13\rangle 1.$ $LL2NVRAM.extensionInProgress$

BY $\langle 3\rangle 3$

$\langle 13\rangle 2.$ UNCHANGED $LL2NVRAM.extensionInProgress$

BY $\langle 3\rangle 2$

$\langle 13\rangle 3.$ QED

BY $\langle 13\rangle 1$, $\langle 13\rangle 2$

$\langle 12\rangle 2.$ $LL2SPCR' \neq BaseHashValue$

$\langle 13\rangle 1.$ $LL2SPCR \in HashDomain$

$\langle 14\rangle 1.$ $LL2SPCR \in HashType$

BY $\langle 2\rangle 1$ DEF $LL2TypeInvariant$

$\langle 14\rangle 2.$ QED

BY $\langle 14\rangle 1$ DEF $HashDomain$

$\langle 13\rangle 2.$ $fakeHash \in HashDomain$

BY $\langle 3\rangle 2$

$\langle 13\rangle 3.$ QED

BY $\langle 10\rangle 1$, $\langle 13\rangle 1$, $\langle 13\rangle 2$, $BaseHashValueUnique$

$\langle 12\rangle 3.$ QED

BY $\langle 12\rangle 1$, $\langle 12\rangle 2$ DEF $LL2NVRAMLogicalHistorySummary$

$\langle 11\rangle 2.$ QED

BY $\langle 11\rangle 1$

$\langle 10\rangle 3.$ QED

BY $\langle 10\rangle 1$, $\langle 10\rangle 2$

Second, we prove that the extension field of the successor history summary from *LL2HistorySummaryIsSuccessor* is a hash whose second argument is the secured input from the *Successor* operator. This follows directly from the definition of *successorHistorySummary*.

$\langle 9\rangle 2.$ $successorHistorySummary.extension =$
$Hash(previousLL2HistorySummary.extension, securedInput)$

BY DEF $successorHistorySummary$, $Successor$, $securedInput$

Third, we prove that the two hashes are unequal. We will use the *HashCollisionResistant* property, along with the fact (which we will prove in a sub-step) that the second arguments to the two hash functions are unequal.

$\langle 9\rangle 3.$ $Hash(LL2SPCR, fakeHash) \neq$
$Hash(previousLL2HistorySummary.extension, securedInput)$

To employ the *HashCollisionResistant* property, we need to prove some types.

$\langle 10\rangle 1.$ $LL2SPCR \in HashDomain$

$\langle 11\rangle 1.$ $LL2SPCR \in HashType$

BY $\langle 2\rangle 1$ DEF $LL2TypeInvariant$

$\langle 11\rangle 2.$ QED

BY $\langle 11\rangle 1$ DEF $HashDomain$

$\langle 10\rangle 2.$ $fakeHash \in HashDomain$

BY $\langle 3\rangle 2$

$\langle 10\rangle 3.$ $previousLL2HistorySummary.extension \in HashDomain$

$\langle 11\rangle 1.$ $previousLL2HistorySummary.extension \in HashType$

$\langle 12\rangle 1.$ $previousLL2HistorySummary \in HistorySummaryType$

BY $\langle 7\rangle 6$

$\langle 12\rangle 2.$ QED

BY $\langle 12\rangle 1$ DEF $HistorySummaryType$

⟨11⟩2. QED
  BY ⟨11⟩1 DEF *HashDomain*
⟨10⟩4. $securedInput \in HashDomain$
  ⟨11⟩1. $securedInput \in HashType$
    ⟨12⟩1. $LL2NVRAM.hashBarrier' \in HashDomain$
      ⟨13⟩1. $LL2NVRAM.hashBarrier' \in HashType$
        BY ⟨2⟩1, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*
      ⟨13⟩2. QED
        BY ⟨13⟩1 DEF *HashDomain*
    ⟨12⟩2. $someInput \in HashDomain$
      ⟨13⟩1. $someInput \in InputType$
        BY ⟨6⟩1
      ⟨13⟩2. QED
        BY ⟨13⟩1 DEF *HashDomain*
    ⟨12⟩3. QED
      BY ⟨12⟩1, ⟨12⟩2, *HashTypeSafe*DEF *securedInput*
  ⟨11⟩2. QED
    BY ⟨11⟩1 DEF *HashDomain*

Then we need to prove that the second arguments to the hash functions are unequal. We will employ the restriction on fake hash values from the definition of *LL2CorruptSPCR*.

⟨10⟩5. $fakeHash \neq securedInput$
  ⟨11⟩1. $\forall fakeInput \in InputType :$
    $fakeHash \neq Hash(LL2NVRAM.hashBarrier', fakeInput)$
    ⟨12⟩1. $\forall fakeInput \in InputType :$
      $fakeHash \neq Hash(LL2NVRAM.hashBarrier, fakeInput)$
      BY ⟨3⟩2
    ⟨12⟩2. UNCHANGED $LL2NVRAM.hashBarrier$
      BY ⟨3⟩2
    ⟨12⟩3. QED
      BY ⟨12⟩1, ⟨12⟩2
  ⟨11⟩2. $someInput \in InputType$
    BY ⟨6⟩1
  ⟨11⟩3. QED
    BY ⟨11⟩1, ⟨11⟩2 DEF *securedInput*
⟨10⟩6. QED

Ideally, this QED step should just read:

BY ⟨10⟩1, ⟨10⟩2, ⟨10⟩3, ⟨10⟩4, ⟨10⟩5, *HashCollisionResistant*

However, the prover seems to get a little confused in this instance. We make life easier for the prover by definining some local variables and hiding their definitions before appealing to the *HashCollisionResistant* assumption.

  ⟨11⟩ $h1a \triangleq LL2SPCR$
  ⟨11⟩ $h2a \triangleq fakeHash$
  ⟨11⟩ $h1b \triangleq previousLL2HistorySummary.extension$
  ⟨11⟩ $h2b \triangleq securedInput$
  ⟨11⟩1. $h1a \in HashDomain$
    BY ⟨10⟩1
  ⟨11⟩2. $h2a \in HashDomain$
    BY ⟨10⟩2
  ⟨11⟩3. $h1b \in HashDomain$
    BY ⟨10⟩3
  ⟨11⟩4. $h2b \in HashDomain$

BY ⟨10⟩4

⟨11⟩5. $h2a \neq h2b$

BY ⟨10⟩5

⟨11⟩6. $Hash(h1a, h2a) \neq Hash(h1b, h2b)$

⟨12⟩ HIDE DEF $h1a$, $h2a$, $h1b$, $h2b$

⟨12⟩1. QED

BY ⟨11⟩1, ⟨11⟩2, ⟨11⟩3, ⟨11⟩4, ⟨11⟩5, *HashCollisionResistant*

⟨11⟩7. QED

BY ⟨11⟩6

⟨9⟩4. QED

BY ⟨9⟩1, ⟨9⟩2, ⟨9⟩3

⟨8⟩2. QED

BY ⟨8⟩1

Second, we prove that the logical history summary cannot be a checkpoint.

⟨7⟩10. $LL2NVRAMLogicalHistorySummary' \neq checkpointedSuccessorHistorySummary$

The extension field of the logical history summary does not equal the base hash value, as we proved above.

⟨8⟩1. $LL2NVRAMLogicalHistorySummary.extension' \neq BaseHashValue$

BY ⟨4⟩1

The extension field of the checkpointed successor does equal the base hash value. This follows from the *CheckpointHasBaseExtensionLemma*.

⟨8⟩2. $checkpointedSuccessorHistorySummary.extension = BaseHashValue$

⟨9⟩1. $successorHistorySummary \in HistorySummaryType$

⟨10⟩1. $previousLL2HistorySummary \in HistorySummaryType$

BY ⟨7⟩6

⟨10⟩2. $someInput \in InputType$

BY ⟨6⟩1

⟨10⟩3. $LL2NVRAM.hashBarrier' \in HashType$

BY ⟨2⟩1, *LL2SubtypeImplicationLemma*DEF *LL2SubtypeImplication*

⟨10⟩4. QED

BY ⟨10⟩1, ⟨10⟩2, ⟨10⟩3, *SuccessorTypeSafe*DEF *successorHistorySummary*

⟨9⟩2. QED

BY ⟨9⟩1, *CheckpointHasBaseExtensionLemma*

DEF *checkpointedSuccessorHistorySummary*

⟨8⟩3. QED

BY ⟨8⟩1, ⟨8⟩2

We thus have a contradiction.

⟨7⟩11. QED

BY ⟨7⟩8, ⟨7⟩9, ⟨7⟩10

⟨6⟩4. QED

BY ⟨6⟩3 DEF *ll1SomeHistoryStateBinding*

We prove the third conjunct within the *nvram* conjunct of the *LL1RestrictedCorruption* action.

⟨5⟩4. $LL1NVRAM' = [$
$historySummary \mapsto LL1NVRAM.historySummary'$,
$symmetricKey \mapsto LL1NVRAM.symmetricKey]$

⟨6⟩1. $LL1NVRAM \in LL1TrustedStorageType$

BY ⟨2⟩1 DEF *LL2Refinement*

⟨6⟩2. $LL1NVRAM' \in LL1TrustedStorageType$

BY ⟨2⟩1 DEF *LL2Refinement*

⟨6⟩3. UNCHANGED $LL1NVRAM.symmetricKey$

⟨7⟩1. UNCHANGED $LL2NVRAM.symmetricKey$

BY ⟨3⟩1 DEF *LL2CorruptSPCR*

$\langle 7\rangle 2.$ QED
BY $\langle 2\rangle 1$, $\langle 7\rangle 1$, *UnchangedNVRAMSymmetricKeyLemma*
$\langle 6\rangle 4.$ QED
BY $\langle 6\rangle 1$, $\langle 6\rangle 2$, $\langle 6\rangle 3$, *LL1NVRAMRecordCompositionLemma*
$\langle 5\rangle 5.$ QED
BY $\langle 5\rangle 1$, $\langle 5\rangle 2$, $\langle 5\rangle 3$, $\langle 5\rangle 4$

Next, we prove the conjunct relating to the RAM. For *LL2CorruptSPCR*, the RAM is unchanged, so we can simply invoke the *UnchangedRAMLemma*.

$\langle 4\rangle 3.$ *LL1RestrictedCorruption*!*ram*
$\langle 5\rangle 1.$ *LL1RestrictedCorruption*!*ram*!*unchanged*
BY $\langle 2\rangle 1$, $\langle 3\rangle 2$, *UnchangedRAMLemma*DEF *LL2CorruptSPCR*
$\langle 5\rangle 2.$ QED
BY $\langle 5\rangle 1$

The remaining variables are unchanged, so we can address each with its appropriate lemma.

$\langle 4\rangle 4.$ UNCHANGED *LL1Disk*
BY $\langle 2\rangle 1$, $\langle 3\rangle 2$, *UnchangedDiskLemma*DEF *LL2CorruptSPCR*
$\langle 4\rangle 5.$ UNCHANGED *LL1AvailableInputs*
BY $\langle 2\rangle 1$, $\langle 3\rangle 2$, *UnchangedAvailableInputsLemma*DEF *LL2CorruptSPCR*
$\langle 4\rangle 6.$ UNCHANGED *LL1ObservedOutputs*
BY $\langle 2\rangle 1$, $\langle 3\rangle 2$, *UnchangedObservedOutputsLemma*DEF *LL2CorruptSPCR*
$\langle 4\rangle 7.$ UNCHANGED *LL1ObservedAuthenticators*
BY $\langle 2\rangle 1$, $\langle 3\rangle 1$, *UnchangedObservedAuthenticatorsLemma*DEF *LL2CorruptSPCR*
$\langle 4\rangle 8.$ QED
BY $\langle 4\rangle 2$, $\langle 4\rangle 3$, $\langle 4\rangle 4$, $\langle 4\rangle 5$, $\langle 4\rangle 6$, $\langle 4\rangle 7$ DEF *LL1RestrictedCorruption*

For the ELSE case, we assume that an extension is not in progress and show that this refines to a stuttering step.

$\langle 3\rangle 4.$ ASSUME $\neg$*LL2NVRAM.extensionInProgress*
PROVE UNCHANGED *LL1Vars*

We first prove that the Memoir-Basic *NVRAM* is unchanged. We do this one field at a time.

$\langle 4\rangle 1.$ UNCHANGED *LL1NVRAM*

The history summary in the Memoir-Basic *NVRAM* is unchanged.

$\langle 5\rangle 1.$ UNCHANGED *LL1NVRAM.historySummary*

The logical history summary in the Memoir-Opt *NVRAM* and *SPCR* is unchanged.

$\langle 6\rangle 1.$ UNCHANGED *LL2NVRAMLogicalHistorySummary*

We reveal the definition of the unprimed logical history summary in the Memoir-Opt *NVRAM* and *SPCR*, given that there is no extension in progress.

$\langle 7\rangle 1.$ *LL2NVRAMLogicalHistorySummary* = [
anchor $\mapsto$ *LL2NVRAM.historySummaryAnchor*,
extension $\mapsto$ *BaseHashValue*]
$\langle 8\rangle 1.$ $\neg$*LL2NVRAM.extensionInProgress*
BY $\langle 3\rangle 4$
$\langle 8\rangle 2.$ QED
BY $\langle 8\rangle 1$ DEF *LL2NVRAMLogicalHistorySummary*

We reveal the definition of the primed logical history summary in the Memoir-Opt *NVRAM* and *SPCR*, given that there is no extension in progress.

$\langle 7\rangle 2.$ *LL2NVRAMLogicalHistorySummary*$'$ = [
anchor $\mapsto$ *LL2NVRAM.historySummaryAnchor*$'$,
extension $\mapsto$ *BaseHashValue*]
$\langle 8\rangle 1.$ $\neg$*LL2NVRAM.extensionInProgress*$'$
$\langle 9\rangle 1.$ $\neg$*LL2NVRAM.extensionInProgress*
BY $\langle 3\rangle 4$

$\langle 9 \rangle 2$. UNCHANGED $LL2NVRAM.extensionInProgress$
BY $\langle 3 \rangle 1$ DEF $LL2CorruptSPCR$
$\langle 9 \rangle 3$. QED
BY $\langle 9 \rangle 1$, $\langle 9 \rangle 2$
$\langle 8 \rangle 2$. QED
BY $\langle 8 \rangle 1$ DEF $LL2NVRAMLogicalHistorySummary$

The history summary anchor in the Memoir-Opt $NVRAM$ is unchanged by a $LL2CorruptSPCR$ action.

$\langle 7 \rangle 3$. UNCHANGED $LL2NVRAM.historySummaryAnchor$
BY $\langle 3 \rangle 1$ DEF $LL2CorruptSPCR$

Since the value of $LL2NVRAMLogicalHistorySummary$ is determined entirely by the history summary in the Memoir-Opt $NVRAM$, and since this value is unchanged, the value of $LL2NVRAMLogicalHistorySummary$ is unchanged.

$\langle 7 \rangle 4$. QED
BY $\langle 7 \rangle 1$, $\langle 7 \rangle 2$, $\langle 7 \rangle 3$
$\langle 6 \rangle 2$. UNCHANGED $LL2NVRAM.symmetricKey$
BY $\langle 3 \rangle 1$ DEF $LL2CorruptSPCR$
$\langle 6 \rangle 3$. UNCHANGED $LL2NVRAM.hashBarrier$
BY $\langle 3 \rangle 1$ DEF $LL2CorruptSPCR$
$\langle 6 \rangle 4$. QED
BY $\langle 2 \rangle 1$, $\langle 6 \rangle 1$, $\langle 6 \rangle 2$, $\langle 6 \rangle 3$, $UnchangedNVRAMHistorySummaryLemma$

The symmetric key in the Memoir-Basic $NVRAM$ is unchanged. This follows directly from the definition of the $LL2CorruptSPCR$ action.

$\langle 5 \rangle 2$. UNCHANGED $LL1NVRAM.symmetricKey$
$\langle 6 \rangle 1$. UNCHANGED $LL2NVRAM.symmetricKey$
BY $\langle 3 \rangle 1$ DEF $LL2CorruptSPCR$
$\langle 6 \rangle 2$. QED
BY $\langle 2 \rangle 1$, $\langle 6 \rangle 1$, $UnchangedNVRAMSymmetricKeyLemma$

Since both fields are unchanged, we can use the $LL1NVRAMRecordCompositionLemma$ to show that the record is unchanged. We first prove the record types, and then we invoke the $LL1NVRAMRecordCompositionLemma$ directly.

$\langle 5 \rangle 3$. $LL1NVRAM \in LL1TrustedStorageType$
BY $\langle 2 \rangle 1$ DEF $LL2Refinement$
$\langle 5 \rangle 4$. $LL1NVRAM' \in LL1TrustedStorageType$
BY $\langle 2 \rangle 1$ DEF $LL2Refinement$
$\langle 5 \rangle 5$. QED
BY $\langle 5 \rangle 1$, $\langle 5 \rangle 2$, $\langle 5 \rangle 3$, $\langle 5 \rangle 4$, $LL1NVRAMRecordCompositionLemma$

The remaining variables are unchanged, so we can address each with its appropriate lemma.

$\langle 4 \rangle 2$. UNCHANGED $LL1RAM$
BY $\langle 2 \rangle 1$, $\langle 3 \rangle 2$, $UnchangedRAMLemma$DEF $LL2CorruptSPCR$
$\langle 4 \rangle 3$. UNCHANGED $LL1Disk$
BY $\langle 2 \rangle 1$, $\langle 3 \rangle 2$, $UnchangedDiskLemma$DEF $LL2CorruptSPCR$
$\langle 4 \rangle 4$. UNCHANGED $LL1AvailableInputs$
BY $\langle 2 \rangle 1$, $\langle 3 \rangle 2$, $UnchangedAvailableInputsLemma$DEF $LL2CorruptSPCR$
$\langle 4 \rangle 5$. UNCHANGED $LL1ObservedOutputs$
BY $\langle 2 \rangle 1$, $\langle 3 \rangle 2$, $UnchangedObservedOutputsLemma$DEF $LL2CorruptSPCR$
$\langle 4 \rangle 6$. UNCHANGED $LL1ObservedAuthenticators$
BY $\langle 2 \rangle 1$, $\langle 3 \rangle 1$, $UnchangedObservedAuthenticatorsLemma$DEF $LL2CorruptSPCR$
$\langle 4 \rangle 7$. QED
BY $\langle 4 \rangle 1$, $\langle 4 \rangle 2$, $\langle 4 \rangle 3$, $\langle 4 \rangle 4$, $\langle 4 \rangle 5$, $\langle 4 \rangle 6$ DEF $LL1Vars$

Both THEN and ELSE cases are proven.

$\langle 3 \rangle 5$. QED

BY $\langle 3\rangle 3$, $\langle 3\rangle 4$

$\langle 2\rangle 12$. QED

BY $\langle 2\rangle 1$, $\langle 2\rangle 2$, $\langle 2\rangle 3$, $\langle 2\rangle 4$, $\langle 2\rangle 5$, $\langle 2\rangle 6$, $\langle 2\rangle 7$, $\langle 2\rangle 8$, $\langle 2\rangle 9$, $\langle 2\rangle 10$, $\langle 2\rangle 11$ DEF *LL1Next*, *LL2Next*

$\langle 1\rangle 4$. QED

Using the *StepSimulation* proof rule, the base case and the induction step together imply that the implication always holds.

$\langle 2\rangle 1$. $\Box[LL2Next]_{LL2Vars} \land \Box LL2Refinement \land \Box LL2TypeInvariant \Rightarrow \Box[LL1Next]_{LL1Vars}$

BY $\langle 1\rangle 3$, *StepSimulation*

$\langle 2\rangle 2$. QED

BY $\langle 1\rangle 1$, $\langle 1\rangle 2$, $\langle 2\rangle 1$ DEF *LL2Spec*, *LL1Spec*, *LL2Refinement*

ACKNOWLEDGMENTS

The authors are deeply in debt to Leslie Lamport, not only for creating the TLA+ language, refinement-based proof methodology, and hierarchical proof structure, but also for his ongoing tutelage and ready assistance during the long process of writing the proofs herein. We also thank Jon Howell for early discussions and advice on the best approach for proving correctness of the Memoir system.

REFERENCES

[1] Kaustuv Chaudhuri, Damien Doligez, Leslie Lamport, and Stephan Merz. A TLA+ proof system. *CoRR*, abs/0811.1914, 2008.

[2] Kaustuv Chaudhuri, Damien Doligez, Leslie Lamport, and Stephan Merz. The TLA+ proof system: Building a heterogeneous verification platform. In *ICTAC*, page 44, 2010.

[3] Kaustuv Chaudhuri, Damien Doligez, Leslie Lamport, and Stephan Merz. Verifying safety properties with the TLA+ proof system. In *IJCAR*, pages 142–148, 2010.

[4] Leslie Lamport. How to write a proof. *American Mathematical Monthly*, 102(7):600–608, August–September 1993.

[5] Leslie Lamport. Refinement in state-based formalisms. Technical Report 1996-001, SRC, December 1996.

[6] Leslie Lamport. *Specifying systems: the TLA+ language and tools for hardware and software engineers.* Addison-Wesley Professional, 2002.

[7] Leslie Lamport. TLA+2: A preliminary guide. http://research.microsoft.com/en-us/um/people/lamport/tla/tla2-guide.pdf, July 2010.

[8] Bryan Parno, Jacob R. Lorch, John R. Douceur, James Mickens, and Jonathan M. McCune. Memoir: Practical state continuity for protected modules. In *IEEE Symposium on Security and Privacy*, May 2011.