

Secrecy in Squirrel and the Post-Compromise Security of a Ratchet

Clément Hérouard

Université de Lorraine, CNRS, Inria, LORIA
Nancy, France
clement.herouard@inria.fr

Adrien Koutsos

Inria
Paris, France
adrien.koutsos@inria.fr

Charlie Jacomme

Université de Lorraine, CNRS, Inria, LORIA
Nancy, France
charlie.jacomme@inria.fr

JosephALLEMAND

Univ. Rennes, CNRS, IRISA
Rennes, France
joseph.lallemand@irisa.fr

Abstract

Ratchets are critical cryptographic protocols deployed in many secure messaging applications such as Signal Messenger, WhatsApp, and Apple’s iMessage PQ3, which aim for strong guarantees such as Post-Compromise Security (PCS). Computer-aided cryptography can be used to increase confidence in security protocols but, until now, has been unable to prove PCS in the computational model for a ratchet due to the complexity of the cryptographic arguments involved. Existing mechanized PCS analyses have been limited to the symbolic model.

We tackle this problem with SQUIRREL, which is well-suited to study such stateful protocols. The task is still a challenge, due to the intricate reasoning required, and because SQUIRREL’s indirect modeling of secrecy has difficulties in scaling to such a complex proof. To address these issues, we develop a novel logical framework in SQUIRREL that allows to reason on secrecy as a first-class notion, and we validate our approach by verifying the PCS of an asymmetric ratchet. This provides the first mechanized computational proof of PCS to date for a ratchet, and possibly for any protocol.

CCS Concepts

• Security and privacy → Logic and verification; Cryptography; • Theory of computation → Logic.

Keywords

Computer-Aided Cryptography, Verification, Security protocols

1 Introduction

Cryptographic protocols are critical systems used to ensure the security of our communications. Provable security [33] establishes strong guarantees on these protocols through mathematical proofs of security. Computer-aided cryptography (CAC) [11] allows to mechanize these proofs, providing an even higher level of confidence. This research area has been quite fruitful, with many frameworks under active development, e.g. [4, 13, 15, 18, 37].

The Squirrel Prover. SQUIRREL [4] is an interactive proof assistant dedicated to the verification of cryptographic designs. It relies on a higher-order probabilistic logic [8] to encode the interaction of an adversary and a protocol, and captures cryptographic arguments using high-level reasoning rules. It provides some support

for proof automation, both for cryptographic arguments – e.g. automated cryptographic reductions [9] – and for generic proofs using SMTs [6]. The tool has been used to verify a number of protocols, including YubiKey [5], the FOO [35] e-voting protocol [10], and post-quantum hybrid KEM combinators and key exchanges [3].

Signal and the Double Ratchet (DR). Signal’s protocol is a widely deployed secure messaging protocol used by Signal Messenger, WhatsApp, and Facebook Messenger. It aims at providing strong confidentiality guarantees in the presence of corruption, through two properties: *Perfect Forward Secrecy (PFS)*, which requires that the compromise of a user should not reveal the content of their past messages; and *Post-Compromise Security (PCS)* [26], which asks that a user can *heal* from corruption, assuming the protocol ran without adversarial interference for sufficiently long.

To achieve these properties, the protocol’s keys are updated frequently in a way that i) prevents future keys from giving any information about past keys (PFS); ii) injects fresh randomness in the key updates to allow for healing after a compromise (PCS). Signal’s protocol has two main sub-protocols: an initial key exchange, with two variants, X3DH and PQXDH, which establishes a shared secret used to derive a starting key; and the *DR*, tasked with regularly updating the keys, itself composed of two sub-ratchets.

The *symmetric ratchet* derives a sequence of *chain keys* from an initial *root key* by repeatedly applying a Key Derivation Function (KDF). This mechanism provides PFS, since the KDF cannot be inverted to recover past chain keys, but does not provide PCS, as the key update is deterministic. The *asymmetric ratchet* constructs a sequence of *root keys*, where each forms a starting point for the symmetric ratchets. Essentially, each root key rk' is obtained by applying a KDF H to the concatenation of the previous root key rk and a fresh shared Diffie-Hellman (DH) secret s_{dh} , i.e. $rk' = H(rk, s_{dh})$. Thus, rk' is secret if any of rk or s_{dh} is secret. That way, even if rk is compromised, secrecy can be recovered provided that the DH exchange producing s_{dh} is not tampered with.

Such ratchets are nowadays a cornerstone of modern secure messaging, typically used in e.g. Signal’s SPQR [28], Matrix’s Olm [32], and Apple’s iMessage PQ3 [2, 40]. Further, ratchets are deployed at scale: WhatsApp is reported to have billions of monthly users [46]. It is therefore a necessity to ensure that ratchets do provide the expected security properties, which has motivated several pen-and-paper cryptographic analyses [1, 17, 23, 25, 27]. However, to our knowledge, no such machine-checked proof has ever been written.

In this work, we seek to obtain machine-checked proofs of PCS in particular, and we thus focus on the asymmetric ratchet, which is the essential component of the protocol that provides this property.

Secrecy notions and PCS. A subtle point arises when studying PCS due to the different ways secrecy can be expressed. Consider the interactive execution $\mathcal{P}^{\mathcal{A}}$ of a protocol $\mathcal{P}$, which produces a secret key, with an adversary $\mathcal{A}$ controlling the network. In cryptographic proofs, we can think of two standard ways of formulating the secrecy of that key, expressed by the experiments below.

$b \leftarrow \{0, 1\}$ $k_0 \leftarrow \mathcal{P}^{\mathcal{A}}$ $k_1 \leftarrow \{0, 1\}^\eta$ $b' \leftarrow \mathcal{A}(k_b)$ return $b = b'$	$k \leftarrow \mathcal{P}^{\mathcal{A}}$ $k' \leftarrow \mathcal{A}()$ return $k = k'$
---	---

Real-or-random secrecy (RoR) requires that the advantage of any adversary $\mathcal{A}$ in the game on the left is negligibly close in η to $\frac{1}{2}$ (i.e. the random guess). In other words, $\mathcal{A}$ must be unable to distinguish the actual key k_0 computed by the protocol from a freshly sampled k_1 . On the other hand, *non-deduction*, on the right, requires that any adversary has a negligible probability of computing k . It is well-known that RoR implies non-deduction, but that the converse is not true: if k is RoR, k with its first bit overwritten to 0 is *not* RoR but remains non-deducible. Many applications require the stronger notion, e.g. an encryption key must be RoR for the IND-CCA assumption to apply. Nevertheless, non-deduction is sometimes sufficient: e.g. when computing a key $k = H(r)$ with H a random oracle, the key k is RoR as long as r is non-deducible.

The crucial observation we make is that RoR secrecy is too strong to be used as an intermediate security notion in all steps of a PCS proof: the compromise of later keys in the ratchet breaks the RoR of past keys, which are consequently only non-deducible.

Example 1.1. Assume that a root key rk is secret, and consider a key update $rk' = H(rk, s_{dh})$ where both the *new* key rk' and the DH value s_{dh} are compromised. This happens in the asymmetric ratchet if our device is compromised while still storing rk' as well as the secret DH key used to compute s_{dh} . Then, the old key rk is *not* RoR secret: the attacker can determine if k_b is the root key rk or a random key k_1 by computing $H(k_b, s_{dh})$ and comparing the result with rk' : if $k_b = rk$, the equality always holds, while it almost never does if $k_b = k_1$. However, the root key rk is still non-deducible.

Thus, while PCS must achieve RoR secrecy for the *encryption* keys derived by the protocol in the end, a cryptographic proof of PCS must either consider a coarser formulation of PCS than necessary, to eliminate such cases, or establish only non-deduction for some of the *root* keys. Note that, as explained above, this weaker secrecy property for root keys is still sufficient to guarantee RoR for the encryption keys derived from them. We found that the seminal pen-and-paper analysis of Signal’s DR [25] incorrectly dealt with this subtle difference, and wrongly claimed RoR secrecy for all root keys, even in scenarios where only non-deduction holds (see Appendix E for more details). It is not surprising that such issues appeared in manual analyses, considering the complexity of the ratchet and of the PCS property. This makes it all the more evident that a *machine-checked proof of the asymmetric ratchet’s PCS in the computational model* is needed, which has remained out of reach of

computer-aided cryptography until now. Our goal in this work is to produce such a proof.

Challenges. The first hurdle is that the asymmetric ratchet is a stateful protocol, with frequent key updates and an unbounded number of messages exchanged in a single session, and that PCS considers complex execution traces with many potential compromises. Out of the current computational verification tools, SQUIRREL has demonstrated the capability to reason on stateful protocols [5] and complex traces, making it a natural choice for our purpose.

A second challenge arises from the fact that developments in SQUIRREL, which is built around the notion of indistinguishability, usually rely on RoR secrecy. While non-deduction can be encoded in SQUIRREL’s logic, this requires many tedious back-and-forth steps between the encoding of non-deduction and the built-in indistinguishability-based security predicates upon which SQUIRREL rules operate. This, in turn, prevents SQUIRREL from scaling to the kind of complex protocols and properties we consider here.

In addition, the ratchet extensively uses KDFs and hash functions, which are more conveniently reasoned about in the Random Oracle Model (ROM) – and indeed, the aforementioned pen-and-paper proof [25] takes place in the ROM. Giving access to the adversary to a random oracle in SQUIRREL, however, is not practical, as noted in [5], and no published protocol analyses in SQUIRREL do so in complex scenarios. Indeed, the ROM tends to complicate proofs, as it often requires proving that some messages cannot have been computed and given as input to the random oracle. This is again an instance of non-deduction, not well-supported by SQUIRREL.

Contributions and thesis. Our thesis is that reasoning on non-deduction natively as a first-class security predicate in SQUIRREL is a powerful and effective approach for capturing cryptographic proofs of complex key update mechanisms, such as the asymmetric ratchet, and for allowing SQUIRREL to scale. We argue for this thesis through the following contributions.

First, we develop a logical framework to reason on non-deduction as a first-class notion. Our logic features a main predicate $\nexists$, where $u \nexists v$ states that it is not possible to compute v from u with more than a negligible probability. We design a sound proof system for $\nexists$, which expresses generic properties of non-deduction, rather than *ad hoc* arguments tailored to the ratchet. We develop our framework as a sub-logic that we embed within SQUIRREL’s higher-order logic [8], though some of our new rules are only shown sound on paper.

Second, we extend our proof system with rules dedicated to reasoning on secrecy in the ROM. These are crucial to conduct a proof in the ROM, as they capture ROM-based cryptographic arguments that go beyond what could be done in SQUIRREL before.

Third, we integrate our logic for non-deduction in SQUIRREL, where it nicely interacts with a pre-existing predicate for deduction, that already has dedicated built-in support [5].

Finally, we use our logic and its implementation to conduct the first mechanized computational proof of PCS for an asymmetric ratchet. We consider a scenario with two agents, an unbounded number of ratchet steps, and a non-adaptive formulation of PCS. Our security proof relies on a non-trivial inductive security argument, and is arguably the most intricate SQUIRREL proof to date.

Limitations. We discuss some limitations of our work. First, we only study PCS for a high-level model of an asymmetric ratchet, and do not cover all the features of a real-world ratchet like the one used in Signal. Notably, we do not model Signal’s session-handling protocol Sesame [43]. Note that this would likely be impossible, as recent work indicates that Sesame does not provide PCS [29]. Second, we only consider two users. As users share no long-term secrets, extending our analysis to a multi-user setting should be a straightforward though likely tedious exercise. Third, we consider a non-adaptive notion of PCS where the length of the execution trace and the compromise scenario cannot be chosen at runtime by the attacker. Recent SQUIRREL developments [7] may provide a way of overcoming this limitation in the future.

Related work (CAC tools). We quickly justify the choice of SQUIRREL as a cryptographic verification tool in this work. Tools based on probabilistic Relational Hoare Logics (pRHLs) [14] (such as EASY-CRYPT [13], VCVIO [47], FCF [44] and CRYPTHOL [15]) are more geared towards verifying cryptographic primitives than protocols. The game-hopping-style tool CRYPTOVERIF [18] is well-suited for protocols (e.g. [19, 42]), but lacks the inductive reasoning on states required for *stateful* protocols. Finally, typing-based approaches (e.g. OWL [36], or the one integrated in SQUIRREL [30]) are not interactive, and it is not clear whether they could capture the fine-grained reasoning on random oracles needed here.

Related Work (Double Ratchet). Our case study differs from prior work in two ways: it is mechanized, which brings stronger guarantees than pen-and-paper proofs, and it is in the computational model, and thus considers a stronger attacker than symbolic analyses. As these two features of our work render a finer comparison to previous analyses irrelevant, we only briefly discuss them below.

The PCS of the DR was analyzed on paper in the seminal work of [25], followed by many other [1, 17, 21, 23, 27]. These analyses, which have been conducted in the computational model, differ in several ways, notably on whether they model the DR in isolation or composed with X3DH/PQXDH, and on whether they are multi or single users. The latest work [27], the most precise to date, is a tight security proof of the DR in the ROM for a multi-user and adaptive setting. While our analysis is less precise, it is *mechanized*.

All prior mechanized analyses of the DR [16, 24, 39, 41] are in the symbolic setting [34], and were based on TAMARIN [45], PROVERIF [20], or DY* [16]. The only study of its PCS property in the symbolic model is in [24], which was written concurrently to our work. This work, which showed PCS for the DR, is more precise than ours in some ways (e.g. it includes the symmetric ratchet and out-of-order messages), but our proof is computational, and thus provides guarantees against a stronger attacker.

Finally, we refer the reader to [27] for a comparison between existing paper proofs, and [24] for mechanized symbolic analyses.

Outline. We present some background and an overview of our approach in Section 2, and our secrecy logic in Section 3. Finally, we apply our approach to verify an asymmetric ratchet in Section 4.

We use colors throughout the paper. To ensure accessibility to color-blind readers, colors only encode redundant information.

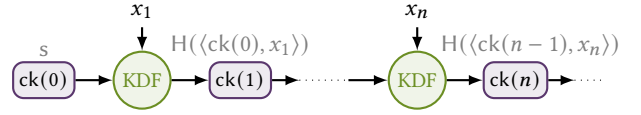


Figure 1: A one-sided asymmetric ratchet chain.

2 Overview

We first recall some background on SQUIRREL, and present an overview of how our secrecy logic operates on a simple example.

We describe in Fig. 1 a simplified asymmetric ratchet in which an agent computes a sequence of *chain keys* $ck(0), ck(1)$, etc. The initial key $ck(0)$ is a randomly sampled value s . Then, for all $n > 0$, the key $ck(n)$ is obtained by applying a KDF H to the concatenation of the previous key $ck(n-1)$ and an input x_n coming from the network. We assume that $H(\cdot)$ is a random oracle, which we model as $H(x) = h(x, k)$ where h is a Pseudorandom Function (PRF) instantiated with a secret key k . The adversary will be given oracle access to $h(\cdot, k)$. Then, for any $n > 0$:

$$ck(n) = h(\langle ck(n-1), x_n \rangle, k)$$

For the sake of simplicity in this example, we do not model the responder’s keys, i.e. we consider a one-sided ratchet, and we consider PFS, which is simpler than PCS. For PFS, we can assume that all the inputs $x_1, x_2, \dots$ are adversarially chosen, as the secrecy of a key $ck(n)$ only rests upon the secrecy of the previous key $ck(n-1)$. Looking ahead, when we consider PCS in Section 4, the security analysis will be more intricate, as the secrecy of $ck(n)$ can also be obtained from the secrecy of the input x_n , which is a DH secret $g^{x \cdot y}$ in a healing update.

2.1 Background on Squirrel

We quickly recall the SQUIRREL framework by presenting some salient features of its logic and execution model.

Language. SQUIRREL describes protocols using a pure probabilistic higher-order λ -calculus [8]. It relies on a simple type system:

$$\tau ::= \tau \rightarrow \tau \mid \tau_b \in \mathbb{B}$$

where $\rightarrow$ denotes a function type and $\mathbb{B}$ is a set of base types (e.g. booleans `bool`, bit-strings `message`). Base types are required to be serializable, i.e. they must correspond to a subset of $\{0, 1\}^*$. Each type τ has associated labels $\text{label}(\tau)$ expressing additional assumptions, e.g. `finite` $\in \text{label}(\tau)$ indicates that τ must be finite.

SQUIRREL uses λ -terms built upon a set of variable symbols $\mathcal{X}$:

$$t ::= \lambda(x : \tau). t \mid x \mid t t$$

where x ranges over $\mathcal{X}$. Probabilities are modeled using arrays of eagerly sampled randomness called *names*. We assume a subset $\mathcal{N} \subset \mathcal{X}$ of *name symbols*, which all have types of the form $\tau_0 \rightarrow \tau_1$ where τ_0 is finite and τ_1 is a base type. Intuitively, a name is an array, indexed by values in τ_0 , of values of type τ_1 that are identically and independently sampled according to a distribution over τ_1 . For example, if `index` is a set $\{1, \dots, N\}$ of agent identities, and if `message` is equipped with the uniform distribution over $\{0, 1\}^\eta$, the name $n : \text{index} \rightarrow \text{message}$ is an array of N bit-strings of length η .

$$\begin{aligned}
\text{frame}(\tau) &= \langle \text{frame}(\text{pred}(\tau)), \text{output}(\tau) \rangle \\
\text{input}(\tau) &= \text{att}(\text{frame}(\text{pred}(\tau))) \\
\text{output}(\tau) &= \begin{cases} \text{empty} & \text{if } \tau = A(i) \\ \text{ck}(\tau) & \text{if } \tau = \text{Corrupt} \\ h(\text{input}(\tau), k) & \text{if } \tau = \text{ROM}(i) \end{cases} \\
\text{ck}(\tau) &= \begin{cases} h(\langle \text{ck}(\text{pred}(\tau)), \text{input}(\tau) \rangle, k) & \text{if } \tau = A(i) \text{ or } \text{Corrupt} \\ \text{empty} & \text{if } \tau = \text{ROM}(i) \\ s & \text{if } \tau = \text{Init} \end{cases}
\end{aligned}$$

In the definitions above, τ is such that $\text{happens}(\tau)$. Implicitly, $f(\text{init}) = \text{empty}$ for all functions f , except for ck .

Figure 2: Instantiation of our execution model on the one-sided asymmetric ratchet.

Execution model. We represent the interaction of a protocol with an adversary adopting SQUIRREL’s standard execution model [5]. For legibility, we only present a simplified encoding on our motivating example. To model the one-sided ratchet of Fig. 1, we consider a sequence of interactions between the adversary and the protocol, where each interaction is characterized by a timestamp τ of type

timestamp ::= Init | A(i) | Corrupt | ROM(i)

where i is a value of a finite type **index**. The timestamp Init denotes the initial timestamp; A(i) denotes the i -th standard key update; Corrupt denotes a key update followed by the corruption of the chain key ck ; and ROM(i) represents the i -th call to the random oracle by the adversary. For simplicity, we consider here a scenario with at most one corruption event, which is why Corrupt is not indexed. An adversary/protocol interaction trace is modeled using:

happens : **timestamp** $\rightarrow$ **bool** **<** : **timestamp** $\rightarrow$ **timestamp** $\rightarrow$ **bool**

happens(τ) indicates that τ is scheduled in the trace; and **<** is a total ordering of scheduled events where $\tau_0 < \tau_1$ states that τ_0 is scheduled before τ_1 . We let $\leq \triangleq < \cup =$. For any scheduled τ , $\text{pred}(\tau)$ is the largest timestamp strictly smaller than τ .

We model the interaction of the adversary with the protocol using the functions **frame**(τ), **input**(τ) **output**(τ) and **ck**(τ), depicted in Fig. 2 and defined by (mutual) recursion over timestamps w.r.t. the order **<**. **frame**(τ) is the sequence of all outputs of the protocol from Init up until τ ; **input**(τ) and **output**(τ) are the input and output of the protocol at timestamp τ ; and **ck**(τ) is the chain key at τ .

The logic. We briefly introduce some useful elements of SQUIRREL’s logic, relying on the asymptotic variant of the logic [8]. The predicate $[\phi]$ states that the boolean term ϕ is overwhelmingly true; and the predicate $u \sim v$ that u and v are computationally indistinguishable. Very roughly, in a model $\mathbb{M}$, which provides an interpretation of all symbols in $\mathcal{X}$, each term t has a probabilistic interpretation $\llbracket t \rrbracket_{\mathbb{M}}^{\eta}$ as a random variable over t ’s type. These predicates are then defined in model $\mathbb{M}$ by

$$\begin{aligned}
\mathbb{M} \models [\phi] & \text{ iff } \Pr(\llbracket \phi \rrbracket_{\mathbb{M}}^{\eta}) \in \text{ow}(\eta) \\
\mathbb{M} \models u \sim v & \text{ iff } \forall \mathcal{A} : \text{PPTM}. |\Pr(\mathcal{A}(\llbracket u \rrbracket_{\mathbb{M}}^{\eta})) - \Pr(\mathcal{A}(\llbracket v \rrbracket_{\mathbb{M}}^{\eta}))| \in \text{negl}(\eta)
\end{aligned}$$

where $\eta \in \mathbb{N}$ is the security parameter; $\text{ow}(\eta)$ and $\text{negl}(\eta)$ are the

set of, respectively, overwhelming and negligible functions. For example, $[\text{input}(\tau) \neq \text{ck}(\tau)]$ states that the probability that the input at time τ is equal to the chain key $\text{ck}(\tau)$ is negligible; and $\text{frame}(\tau), \text{ck}(\tau) \sim \text{frame}(\tau), k_{\text{fresh}}$ states that the protocol guarantees the RoR secrecy of the key $\text{ck}(\tau)$, by requiring its indistinguishability from a fresh random key k_{fresh} , even to an adversary who can see all messages sent over the network by the protocol.

SQUIRREL uses a two-level logic. The *local* level operates below the $[\cdot]$ predicate, and uses the quantifiers and connectives $\forall, \exists, \Rightarrow, \wedge, \vee$. The *global* level operates above $[\cdot]$, and uses the quantifiers and connectives $\tilde{\forall}, \tilde{\exists}, \tilde{\Rightarrow}, \tilde{\wedge}, \tilde{\vee}$. For example,

$[\phi \vee \psi]$ states $\Pr(\llbracket \phi \rrbracket \text{ or } \llbracket \psi \rrbracket)$ is overwhelming;

$[\phi] \tilde{\vee} [\psi]$ states one of $\Pr(\llbracket \phi \rrbracket)$ or $\Pr(\llbracket \psi \rrbracket)$ is overwhelming,

which are not equivalent (take $\phi \triangleq b$ a random coin flip and $\psi \triangleq \neg b$).

Additional global level atoms restrict the possible values of a term: **const**(u) states that u is a constant, whose value is deterministic and independent of the security parameter η ; and **adv**(u) states that u can be computed in polynomial time by the adversary.

SQUIRREL’s user syntax. We now quickly present some concrete syntax of the tool, which is useful throughout the paper – see [31] for a more precise description of SQUIRREL’s syntax.

let $f(x:\text{int}) = t$ lemma $\text{lem}(x:\text{int}) : \phi$	name $n : \text{int} \rightarrow \text{message}$ global lemma $\text{lem}'(y:\text{int}[\text{const}]) : \Phi$
--	---

On the left, we declare a function f with argument x and body t . On the right, we declare a name symbol n . We provide two examples of lemma declarations at the bottom, where ϕ is a local formula using constructs $\forall, \Rightarrow$, etc., and Φ is a global formula using constructs $\tilde{\forall}, \tilde{\Rightarrow}$, etc. The lemma lem corresponds to the formula $[\forall x. \phi]$, while lem' corresponds to $\tilde{\forall} y. \text{const}(x) \Rightarrow \Phi$. Remark the shorthand for restricted quantification using the **const** tag attached to y .

2.2 A Secrecy Predicate

We model non-deduction in SQUIRREL using a novel predicate $u \not\leq s$, which states that s cannot be *efficiently* computed from u :

$$u \not\leq s \triangleq \tilde{\forall} f. \text{adv}(f) \Rightarrow [f(u) \neq s]$$

The atom $u \not\leq s$ states that $f(u)$ is overwhelmingly different from s for any function f that can be efficiently computed by the adversary – “efficiently” is captured by the **adv**(f) predicate from [8].

Our predicate is compositional as the quantification over f can be instantiated by any adversarial computation *within the logic*. In contrast, consider the formula $[\text{input}(\tau) \neq s]$: a meta-level argument shows that it states the secrecy of s , as **input**(τ) models an arbitrary adversarial computation using knowledge **frame**($\text{pred}(\tau)$). However, this formula does not entail e.g. $[\pi(\text{input}(\tau)) \neq s]$ with π the projection function only keeping the first η bits of its input. This compositionality is critical to *reason* on secrecy using our logic.

We model PFS using this new non-deduction predicate:

let $\text{safe}(\tau : \text{timestamp}) = \forall \tau', \tau' \leq \tau \Rightarrow \tau' \neq \text{Corrupt}$. global lemma PFS($\tau, \tau' : \text{timestamp}[\text{const}]$): $[\text{happens}(\tau, \tau')] \Rightarrow [\text{safe}(\tau) \Rightarrow (\text{frame}(\tau') \not\leq \text{ck}(\tau))]$
--

We consider that a timestamp τ is *safe*, which is captured by the **safe**(τ) boolean function, if no corruption occurred before τ was scheduled in the trace. PFS requires that the key computed at any safe timestamps τ remains secret at any point τ' in the execution,

in the sense that $\text{ck}(\tau)$ cannot be efficiently computed from the sequence $\text{frame}(\tau')$ of all protocol messages up to timestamp τ' .

2.3 Proving Secrecy

We give a high-level overview of how our secrecy logic operates by sketching the proof of PFS for the one-sided ratchet. The full SQUIRREL model can be found in our artifact [38] at `Models/motivating-example.sp`. The proof is split into the following three steps:

- (1) (*non-collision*) there are no collisions between distinct and safe chain keys.
- (2) (*simulation*) the frame of the protocol can be simulated from a set of leaked values leakage. Thus, it is sufficient to prove the secrecy of $\text{ck}(\tau)$ w.r.t. leakage.
- (3) (*secrecy*) $\text{ck}(\tau)$ is secret w.r.t. leakage.

Note that steps (1) and (2) are independent and can be reordered.

Non-collision. We prove the absence of collisions between safe chain keys by proving that $[\text{ck}(\tau) \neq \text{ck}(\tau')]$, for any distinct τ and τ' that characterize a safe key update. These are the timestamps of the form Init or $A(i)$ – indeed, the key is not updated when $\tau = \text{ROM}(i)$, and Corrupt is not safe. This yields the lemmas:

lemma $\text{NoCollision}_{\text{Init}}$ (i :index): $\text{happens}(A(i)) \Rightarrow \text{ck}(A(i)) \neq s$.
lemma NoCollision_A (i :index): $A(i) < A(j) \Rightarrow \text{safe}(A(j)) \Rightarrow \text{ck}(A(i)) \neq \text{ck}(A(j))$.

These lemmas are proved using the collision-resistance and pre-image resistance of the hash, as well as information-theoretic arguments. (See [Appendix B](#) for details.)

Simulation. We show that $\text{frame}(\tau')$ can be simulated using a small number of leaked values leakage. We capture this fact by introducing a *deduction* predicate:

$$u \triangleright v \triangleq \exists (f : \tau_0 \rightarrow \tau_1). \text{adv}(f) \wedge [f(u) = v]$$

Deduction is the positive counterpart to the secrecy predicate $\not\vdash$, and requires that there exists an efficient function f that can compute v from u . We do not take credit for this formalization of deduction, as a similar notion has been introduced in [5] – although our exploration of the relations between $\triangleright$ and $\not\vdash$ is novel.

We use $\triangleright$ to state our simulation property. Looking back at [Fig. 2](#), we see that simulating the protocol frame amounts to simulating the sequence of outputs, which can be done using the random oracle the adversary has access to; and the corrupted key $\text{ck}(\text{Corrupt})$, if Corrupt was scheduled. Let $\text{leakage} = (\text{o}_h, C)$ where

$$\text{o}_h \triangleq \lambda x. h(x, k) \quad C \triangleq \text{if happens}(\text{Corrupt}) \text{ then } \text{ck}(\text{Corrupt})$$

We prove by induction over τ that:

global lemma $\text{simulate}(\tau : \text{timestamp}[\text{const}])$:
 $[\text{happens}(\tau)] \Rightarrow (\text{leakage} \triangleright \text{frame}(\tau)).$

Deduction $\triangleright$ and non-deduction $\not\vdash$ interact in interesting ways – see [Section 3.5](#) for more details. For now, we simply observe that:

$$\forall u, u', v. (u' \triangleright u) \Rightarrow (u' \not\vdash v) \Rightarrow (u \not\vdash v).$$

Intuitively, to prove that v cannot be deduced from u , it is sufficient to prove that it cannot be deduced from some u' that can deduce u . With this property and the simulation lemma, we reduce the secrecy of $\text{ck}(\tau)$ w.r.t. $\text{frame}(\tau')$ to the secrecy of $\text{ck}(\tau)$ w.r.t. leakage.

Secrecy. Finally, we must prove that $\text{ck}(\tau)$ is secret w.r.t. leakage.

global lemma $\text{secrecy}(\tau : \text{timestamp}[\text{const}])$:
 $[\text{happens}(\tau)] \Rightarrow [\text{safe}(\tau)] \Rightarrow (\text{leakage} \not\vdash \text{ck}(\tau)).$

As a first step, we prove that the chain key $\text{ck}(\tau)$ is secret when the leakage is restricted to be only the random oracle o_h .

global lemma $\text{secrecy}_{\text{base}}(\tau : \text{timestamp}[\text{const}])$:
 $[\text{happens}(\tau)] \Rightarrow [\text{safe}(\tau)] \Rightarrow (\text{o}_h \not\vdash \text{ck}(\tau)).$

The key $\text{ck}(\tau)$ is a sequence of nested hashes $h(\dots h(s, k), \dots, k)$. The initial chain key s is clearly secret w.r.t. o_h , as it is freshly sampled. However, we note that s is not RoR: as in [Example 1.1](#), the attacker can distinguish s from a fresh secret by using the random oracle to recompute the chain of hashes up to the compromise point and comparing the result with the compromised value. However, s is non-deducible, and we would like to propagate that secrecy to the later keys. Unfortunately, applying computations to a secret value – even a cryptographic hash – does not preserve secrecy. For example, although $h(s, k) \not\vdash s$, it is clear that $h(s, k) \not\vdash h(s, k)$ does not hold. Fortunately, we can show that h can soundly be applied on the right-hand side of $u \not\vdash v$, under the additional condition that v has never been hashed. Roughly, we establish the reasoning rule:

$$\frac{\text{o}_h, u \not\vdash v \quad [\psi_{\text{PRF}, \text{msg}}^{v, k}(u, v)]}{\text{o}_h, u \not\vdash h(v, k)} \quad (1)$$

where $\psi_{\text{PRF}, \text{msg}}^{v, k}(u, v)$ is a formula ensuring that v was never hashed using key k in (u, v) , except with negligible probability. This kind of proof obligation is usual in SQUIRREL, e.g. it appears in its indistinguishability rule for PRFs [8].

We can now prove the $\text{secrecy}_{\text{base}}$ lemma by induction over the number of nested hashes. Its initialization stems from the earlier observation that s is secret. The difficulty lies in the induction step where, assuming that $\text{ck}(\text{pred}(\tau))$ is secret, we must prove that

$$\begin{aligned} \text{o}_h \not\vdash \text{ck}(\tau) \quad & \text{where } \text{ck}(\tau) = h(m, k) \\ & \text{and } m = \langle \text{ck}(\text{pred}(\tau)), \text{input}(\tau) \rangle \end{aligned}$$

Using the PRF rule in [Eq. \(1\)](#), we need only to show that m is secret – we know that m cannot have been hashed in the past thanks to the non-collision lemmas. We deduce the secrecy of m from the induction hypothesis $\text{o}_h \not\vdash \text{ck}(\text{pred}(\tau))$, using again an interaction between $\not\vdash$ and $\triangleright$:

$$\forall u, v, v'. (v \triangleright v') \Rightarrow (u \not\vdash v') \Rightarrow (u \not\vdash v),$$

or, in other words: to prove v secret, it suffices to show that a secret value v' can be computed from v . With this property, we conclude the induction: the secrecy of $\text{ck}(\text{pred}(\tau))$ entails that of m , since $m \triangleright \text{ck}(\text{pred}(\tau))$ – this is straightforward as $m = \langle \text{ck}(\text{pred}(\tau)), \text{input}(\tau) \rangle$.

We are now ready to prove the main secrecy lemma. Recall that $\text{leakage} = (\text{o}_h, C)$. To prove that $\text{leakage} \not\vdash \text{ck}(\tau)$, we are going to show that the leaked corrupted key C brings no additional useful knowledge to the adversary. This will reduce $\text{leakage} \not\vdash \text{ck}(\tau)$ to $\text{o}_h \not\vdash \text{ck}(\tau)$, which we already know thanks to $\text{secrecy}_{\text{base}}$. To prove that C brings no useful knowledge, we observe that $C = h(v, k)$ where $v = \langle \text{ck}(\text{pred}(\text{Corrupt})), \text{input}(\text{Corrupt}) \rangle$.

In essence, the argument should be that C is a hashed value, which is the same as a random value, since we are in the ROM

– and therefore useless to the adversary. However, making this intuitive argument precise requires careful reasoning. We need to understand when the following implication holds:

$$(o_h, u \not\vdash t) \Rightarrow (o_h, u, h(v, k) \not\vdash t)$$

Obviously, it cannot hold unconditionally, e.g. if $t = h(v, k)$. This raises the following question: *are there conditions under which the hash $h(v, k)$ brings no useful information in a secrecy goal?* We answer positively, and show that this is the case if v is itself secret and if v was never hashed in the past. Intuitively, the hash $h(v, k)$ then behaves as a fresh random sampling, since $h(\cdot, k)$ is a random oracle. Since a fresh random value brings no useful information, it can be safely discarded. Roughly, this is captured by the rule:

$$\frac{o_h, u \not\vdash t \quad o_h, u \not\vdash v \quad [\psi_{\text{PRF}, \text{msg}}^{v, k}(u, v, t)]}{o_h, u, h(v, k) \not\vdash t}$$

When proving our secrecy lemma, the two secrecy premises:

$$o_h \not\vdash v \quad \text{and} \quad o_h \not\vdash \text{ck}(\tau)$$

are direct consequences of $\text{secrecy}_{\text{base}}$. The absence of collision between $\text{ck}(\tau)$ and previously hashed values follows from the non-collision lemmas and from the $\text{secrecy}_{\text{base}}$ lemma (to show that previous calls to the ROM must be different from v).

Conclusion on the example. We have presented the main steps of a PFS proof for a simple ratchet, the way they rely on the notion of non-deduction, and a few simple proof rules to reason about it. Although much more complex, the proof of PCS for the actual asymmetric ratchet in Section 4 follows the same ideas at its core.

3 A Logical Framework for Secrecy

We now formally introduce the deduction and non-deduction predicates, and present a proof system to reason about them. From a theoretical point of view, this framework is presented in the context of the higher-order logic of [8], and its development does not require modifying SQUIRREL’s logical foundations. We discuss its implementation in SQUIRREL at the end of this section.

3.1 Secrecy Predicates

Recall from Section 2.2 and Section 2.3 that the non-deduction and deduction predicates are formally expressed as global formulas:

- $u \not\vdash v \triangleq \tilde{\forall} f. \text{adv}(f) \Rightarrow [f(u) \neq v]$
- $u \triangleright v \triangleq \exists f. \text{adv}(f) \wedge [f(u) = v]$

Note that the predicate $\text{adv}(\cdot)$ is only defined for terms of order at most 2, and therefore u and v must in turn be of order at most 1 [7]. For convenience, when u or v are tuples, e.g. $u = (u_1, u_2, \dots, u_n)$, we will write the formula directly as $u_1, \dots, u_n \not\vdash v$.

Preliminary remarks. Both predicates are mutually exclusive: if $u \triangleright v$, there exists an adversarial f that computes v from u with an overwhelming probability, and thus *a fortiori* with a non-negligible one, meaning that $u \not\vdash v$ cannot also hold. Note, however, that the two predicates are not each other’s negation: indeed, it can happen that neither holds, if there only exist functions that compute v with a non-negligible, but non-overwhelming, probability.

Example 3.1. Take a constant $u = 0$, and a coin toss $v \leftarrow \{0, 1\}$. Clearly $u \not\vdash v$ does not hold, e.g. by taking f to be the function that always returns 0: f computes v with probability $\frac{1}{2}$, which is not negligible. But, since v is a random bit drawn independently of adversarial randomness, no adversary can do significantly better than that, and thus $u \triangleright v$ does not hold either.

We also remark that the non-deducibility of (v, w) does not imply that v and w are both non-deducible – unsurprisingly, it could be that e.g. only v is. More interestingly, it does not even imply that at least one of them is, i.e. $u \not\vdash (v, w) \Rightarrow u \not\vdash v \wedge u \not\vdash w$.

Example 3.2. The term $u = \text{if } b \text{ then } n \text{ else } m$ is either n or m with probability $\frac{1}{2}$, depending on a random bit b . It is clear that $u \not\vdash n$ does not hold, since u directly gives the value of n half the time, and similarly for $u \not\vdash m$. However, if n and m are two names over $\{0, 1\}^\eta$, it is true that $u \not\vdash \langle n, m \rangle$: indeed, though u always reveals one of n, m , it never reveals both at the same time, and the pair $\langle n, m \rangle$ can only be computed with negligible probability.

Thus, we cannot express the secrecy of multiple values by putting them in a tuple on the right of $\not\vdash$, and must instead write a formula combining several instances of $\not\vdash$.

Proofsystem. We propose a novel proof system for secrecy, which directly manipulates deduction and non-deduction as first-class citizens. We present our proof system as inference rules, with the standard interpretation: the validity of the conclusion must follow from that of the premises. It comprises core rules (Section 3.2) that express fundamental properties of the predicates, and rules for probabilistic properties or cryptographic assumptions (Section 3.3 and 3.4). Some rules can be derived within the logic itself, unfolding the definition of $\triangleright$ and $\not\vdash$ and using SQUIRREL’s indistinguishability and overwhelming truth rules [8]. Others, in particular for oracles and PRFs, go beyond what can be proved within the logic, and are argued for on paper. Nevertheless, all rules use the standard syntax and semantics of SQUIRREL from [8]: only (some) soundness arguments are outside the logic. As usual in this kind of approach, we do not aim to be complete, and argue for generality through practical case-studies. The rest of the section presents a selection of rules. Missing rules and proofs can be found in Appendix C.

3.2 Core Rules

Fig. 3 displays a set of core rules expressing fundamental properties of the deduction and non-deduction predicates.

Rules NEG1 and NEG2 express that, as noted earlier, $\triangleright$ and $\not\vdash$ are incompatible. Rule INEQ follows a similar idea: if v is non-deducible from u but w is deducible, then v and w cannot be equal with more than a negligible probability – otherwise, deducing w would too often let the attacker obtain v . Rule TRANS states the transitivity of $\triangleright$. On the other hand, $\not\vdash$ is not transitive: by just taking two independent random samplings, i.e. two names n, m , it is clear that $n \not\vdash m$ and $m \not\vdash n$, but not $n \not\vdash n$. Rules WEAKL and WEAKR are weakening rules for $\not\vdash$, allowing to replace respectively the left- and right-hand sides of a non-deduction with deducible terms: if w cannot be deduced from u , then *a fortiori* it cannot be deduced from any v deducible from u , and any v from which w could be deduced is non-deducible from u as well. Rule RW-EQUIV expresses an interaction between $\not\vdash$ and $\sim$, transferring non-deduction properties through

NEG1 $\frac{u \triangleright v}{\neg(u \triangleright v)}$	NEG2 $\frac{u \triangleright v}{\neg(u \not\triangleright v)}$	INEQ $\frac{u \not\triangleright v \quad u \triangleright w}{[v \neq w]}$
TRANS $\frac{u \triangleright v \quad v \triangleright w}{u \triangleright w}$	WEAKL $\frac{u \not\triangleright w \quad u \triangleright v}{v \not\triangleright w}$	WEAKR $\frac{u \not\triangleright w \quad v \triangleright w}{u \not\triangleright v}$
RW-EQUIV $\frac{u_2 \not\triangleright v_2 \quad u_1, v_1 \sim u_2, v_2}{u_1 \not\triangleright v_1}$	RW-ORACLE $\frac{u, o_2 \not\triangleright v \quad \tilde{v}m. (u, o_1 \triangleright m) \Rightarrow [o_1(m) = o_2(m)]}{u, o_1 \not\triangleright v}$	
PC-ORACLE $\frac{u, o' \not\triangleright v \quad u, o' \not\triangleright m}{u, o \not\triangleright v}$	where $o' = \lambda x. \text{if } x \neq m \text{ then } o(x) \text{ else } w$	

Figure 3: Core rules for $\triangleright$ and $\not\triangleright$.

an equivalence. More precisely, when proving $u_1 \not\triangleright v_1$, we may rewrite both sides of $\not\triangleright$ according to an equivalence $u_1, v_1 \sim u_2, v_2$. Intuitively, if v_2 cannot be deduced from u_2 , then neither can v_1 from u_1 : otherwise, running the computation violating $u_1 \not\triangleright v_1$ would distinguish (u_1, v_1) from (u_2, v_2) .

The last two rules of Fig. 3 allow rewriting oracles – represented by first-order terms – appearing on the left of $\not\triangleright$. Rule **RW-ORACLE** states that, when proving $u, o_1 \not\triangleright v$, the oracle o_1 can be replaced with a different oracle o_2 if o_1 and o_2 have similar behavior. More precisely, o_1 and o_2 must return the same value (overwhelmingly) when called on any message that can be deduced from u, o_1 : indeed, an adversary trying to compute v from u, o_1 may only call o_1 on values that it can compute from the very same inputs (u, o_1) . Interestingly, the alternative rule in which we instead require that the input m can be deduced from (u, o_2) is also sound. A use case of **RW-ORACLE** is to puncture an oracle, by changing its output on a given point m to a different value w . The second premise can then be discharged by proving that m itself is secret: this corresponds to rule **PC-ORACLE**. Interestingly, if m and v are the same, the two resulting premises are identical: when proving the secret of a given value, any oracle can freely be punctured on that value.

All the rules of Fig. 3 – except **RW-ORACLE** and **PC-ORACLE** – are proved directly in **SQUIRREL** (proofs available at [38]). These last two, and additional rules, are proved in Appendix C.

3.3 Probabilistic and Cryptographic Rules

Fig. 4 features rules that express the freshness of names or are consequences of cryptographic assumptions.

These rules can be seen as a rephrasing in terms of non-deduction of existing proof rules in **SQUIRREL**'s for $[\cdot]$. Similarly to those, they are written using formulas ψ : that are derived from the syntax of the terms considered, and express side-conditions, e.g. ensuring that a given name does not appear in a term. We refer the reader to [8] for the precise definition of these formulas.

The rule **FRESHL** states that a fresh name n is not useful when attempting to deduce any message: indeed, if n is fresh, i.e. is a random sampling independent of all other messages considered,

FRESHL $\frac{u \not\triangleright v \quad [\psi_{\text{fresh}}^n(u, v)]}{u, n \not\triangleright v}$	CDH $\frac{[\psi_{\text{CDH}}^{g,a}(u)] \quad [\psi_{\text{CDH}}^{g,b}(u)]}{u \not\triangleright g^{a \cdot b}}$
PRF-SIMPLE $\frac{u \not\triangleright v \quad [\psi_{\text{PRF,key}}^k(u, v, m)] \quad [\psi_{\text{PRF,msg}}^{m,k}(u, v, m)]}{u, h(m, k) \not\triangleright v}$	PRF-R $\frac{u, o_h^k \not\triangleright m \quad [\psi_{\text{PRF,key}}^k(u, m)] \quad [\psi_{\text{PRF,msg}}^{m,k}(u)]}{u, o_h^k \not\triangleright h(m, k)}$
PRF-L-X $\frac{u, o_h^k \not\triangleright v \quad [\psi_{\text{PRF,key}}^k(u, v, m)] \quad [\psi_{\text{PRF,msg}}^{m,k}(u, v)] \quad C_X}{u, o_h^k \not\triangleright h(m, k)}$	

The last two PRF rules use the oracle $o_h^k = \lambda x. h(x, k)$. All rules but **FRESHL** additionally require, in a condition omitted here, terms to be polytime simulatable (see Fig. 11 for the unabridged rules).

Figure 4: Some probabilistic and cryptographic rules for $\not\triangleright$.

then an adversary could simply simulate it by drawing a random value on its own, and thus does not need it. The syntactic side-condition $\psi_{\text{fresh}}^n(u, v)$ ensures that n is fresh.

Several rules from **SQUIRREL**'s existing proof system, expressing cryptographic assumptions on primitives, can be cast as non-deduction rules with little change. As an example, we present the case of the **CDH** rule, which expresses the identically named CDH assumption on a cyclic group with a generator g : if a, b are exponents drawn at random, then from g^a and g^b it is not possible to compute $g^{a \cdot b}$ with good probability. The rule thus concludes $u \not\triangleright g^{a \cdot b}$, under a condition $\psi_{\text{CDH}}^{g,a}(u)$ which expresses that a only appears in the term u in the subterm g^a , and similarly for b .

Another rule captures the intuition that, as the output of a PRF is indistinguishable from a random value, it does not help in deducing some message v . Our first formulation of the PRF assumption as non-deduction can be obtained from the existing **SQUIRREL** indistinguishability PRF rule [8]:

$$\frac{[\psi_{\text{PRF,msg}}^{m,k}(u, v, m)] \quad [\psi_{\text{PRF,key}}^k(u, v, m)]}{(u, h(m, k)), v \sim (u, n), v}$$

which establishes that a hash $h(m, k)$ is indistinguishable from a fresh nonce, provided that the input m has not been hashed in (u, v, m) , and that the key k is not used for any other purpose – as before, these two conditions are expressed by ψ formulas. By applying the **RW-EQUIV** and **FRESHL** rules shown earlier, we obtain the **PRF-SIMPLE** rule of Fig. 4.

We present in Appendix C.2 rules for other assumptions, such as EUF-CMA and GDH, as well as a right-hand side variant of **FRESHL**.

3.4 The Case of PRF

The cryptographic rules in the previous section are mere reformulations of existing **SQUIRREL** rules in terms of non-deduction. In contrast, we found that, to reason finely about non-deduction

involving hashes produced by PRFs, dedicated rules that do not directly follow from pre-existing ones were necessary.

Indeed, while **PRF-SIMPLE** is sound, we found it restrictive while verifying the ratchet (Section 4). First, this rule relies on the RoR secrecy of the hash $h(m, k)$ to establish non-deduction, and thus does not use fully the latitude afforded by that notion. As a consequence, as we will explain, it only applies when m has not been hashed before, which is too rigid for some scenarios. Second, it does not support the random oracles that are required by the ROM. We present below several rules addressing these limitations. Even though this work is in the asymptotic setting, we also present several rule variants with improved attacker's advantage loss, paving the way for results in the concrete setting in the future.

Ignoring within-hash occurrences. The **PRF-SIMPLE** rule says nothing about cases where, to produce m , an adversary has already hashed m itself: the hash is then distinguishable from random, but may still be non-deducible.

Example 3.3. Consider an attacker who constructs m by asking for the hash of two nonces n and n' , fresh in u and v , and picks the one producing the larger hash (when seen as integers in binary): $m = \text{if } h(n, k) > h(n', k) \text{ then } n \text{ else } n'$. The hash $h(m, k)$ is then not indistinguishable from random, as its distribution is skewed towards higher values. Accordingly, the **PRF-SIMPLE** rule does not apply: $\psi_{\text{PRF,msg}}^{m,k}(u, v, m)$ is not satisfied since the message m , which is either n or n' , is hashed in m itself. However, as n, n' are fresh in u and v , the hash $h(m, k)$ basically amounts to the adversary getting one of two fresh random values, which does not give more deductive power: actually, it is true that $u \not\vdash v$ implies $u, h(m, k) \not\vdash v$.

To eliminate a hash on the left-hand side in such a case, we update the **PRF-SIMPLE** rule by no longer applying $\psi_{\text{PRF,msg}}^{m,k}(\cdot)$ to m . This means that we can ignore how m is constructed (as long as it is done in polynomial time): it may be chosen depending on its own hash, and in that case the hash will not be indistinguishable from random, but that does not matter. This worsens the bound on the attacker's advantage, but enables more expressivity.

Managing the hash oracle. Another limitation of **PRF-SIMPLE** is that it cannot deal with cases where the adversary is directly given access to a hashing oracle $\text{oh}^k \triangleq \lambda x. h(x, k)$ on the left-hand side of $\not\vdash$. Such oracles are desirable in general when reasoning about PRFs, and are vital to the structure of our proof, as explained earlier. In such a case, the formula $\psi_{\text{PRF,msg}}^{m,k}(u, v, m)$ would detect that a hash is computed in the oracle, and would contain the conjunct $\forall x. x \neq m$, which is of course not provable.

Therefore, we need to make it so that $\psi_{\text{PRF,msg}}^{m,k}(\cdot)$ is no longer applied to the oracle. For the right-hand side rule which proves a hash secret, this is rather straightforward: we simply require that the attacker cannot compute m , in which case it cannot get the target hash from the oracle. This yields the **PRF.R** rule in Fig. 4.

The left-hand-side case is a bit more subtle: we note that providing the attacker with a fresh hash $h(m, k)$ in the context of a hashing oracle only amounts to giving it an equality testing oracle $\lambda x. x = m$. Indeed, all the attacker can do with the fresh hash is using the hashing oracle to compute hash values and test whether it gets $h(m, k)$. This is similar to the wide-spread case where a random

oracle applied to a DH exponentiation effectively gives the attacker a DDH oracle. Informally, under the right freshness conditions:

$$((\text{oh}_h^k, u, \lambda x. x = m) \not\vdash v) \stackrel{\sim}{\Rightarrow} (\text{oh}_h^k, u, h(m, k) \not\vdash v)$$

We propose different conditions to then eliminate the input $\lambda x. x = m$, leading to three variants of the rule **PRF.L-X**, displayed in Fig. 4, which is parameterized by an extra condition C_X .

- **PRF.L-EQ** : if we require $C_{\text{eq}} \triangleq \text{oh}_h^k, u \triangleright (\lambda x. x = m)$, then the equality oracle does not give the attacker any information it could not already compute (see **WEAKL**).
- **PRF.L-SEC** : if we require $C_{\text{sec}} \triangleq \text{oh}_h^k, u \not\vdash m$, then the attacker will never be able to input m to $(\lambda x. x = m)$, which is then simply equivalent to $(\lambda x. \perp)$, and thus useless.
- **PRF.L** : we may also set $C_T = \top$ and drop the test oracle. Intuitively, an adversary can simulate the equality oracle by guessing which of the polynomially many distinct values it will call the oracle on will produce a positive answer. This variant adds a multiplicative polynomial factor to the advantage bound.

All these variants are not derived from **SQUIRREL**'s existing rules, since the latter only apply when the hash considered is RoR, which is not the case here. Instead, we prove their soundness from scratch by reductions to the usual PRF game (see Appendix C.3).

3.5 Implementation

Prior to this work, **SQUIRREL** lacked a way to reason about secrecy at a logical level. However, the tool had already been extended with support for deduction in [5], taking the form of an internal deduction engine used to automatically simplify equivalences – more precisely, this mechanism targets *bi*-deduction, an alternative notion which can be repurposed for deduction.

While this machinery internally relied on deduction, the deduction predicate was not exposed to the user in the logic. The reason is that [5] predates the inclusion of higher-order constructions in **SQUIRREL** [8], which we use to quantify over an adversarial computation when defining $\triangleright$ and $\not\vdash$. In contrast, we can now define both predicates in **SQUIRREL**'s user-space:

$$\begin{aligned} \text{predicate } (\not\vdash) [\alpha \beta] (u : \alpha, v : \beta) &= \tilde{\forall} f : (\alpha \rightarrow \beta)[\text{adv}], [f(u) \neq v]. \\ \text{predicate } (\triangleright) [\alpha \beta] (u : \alpha, v : \beta) &= \tilde{\exists} f : (\alpha \rightarrow \beta)[\text{adv}], [f(u) = v]. \end{aligned}$$

We have produced a library for **SQUIRREL** containing these definitions, as well as the rules of Fig. 3 as lemmas (for those proved in **SQUIRREL**). However, some rules would not be very usable written that way: for instance, for **RW-ORACLE**, **SQUIRREL**'s lemmas do not give the user a concise way of pointing out which part of the left-hand side is the oracle, and would instead expect the oracle explicitly as a parameter. To alleviate this issue, we have extended **SQUIRREL** with built-in tactics for **RW-EQUIV**, **RW-ORACLE**, and **PC-ORACLE**, making them easier to apply for users. As the rest of **SQUIRREL**'s code, the implementation of these tactics is not itself formally verified, and they thus become part of **SQUIRREL**'s trusted code base – there is a trade-off here between usability and trust. In contrast, the rules in Fig. 4 cannot be expressed as lemmas, as the ψ conditions cannot be written within **SQUIRREL**. Therefore, we implemented them directly as new built-in tactics – except for a

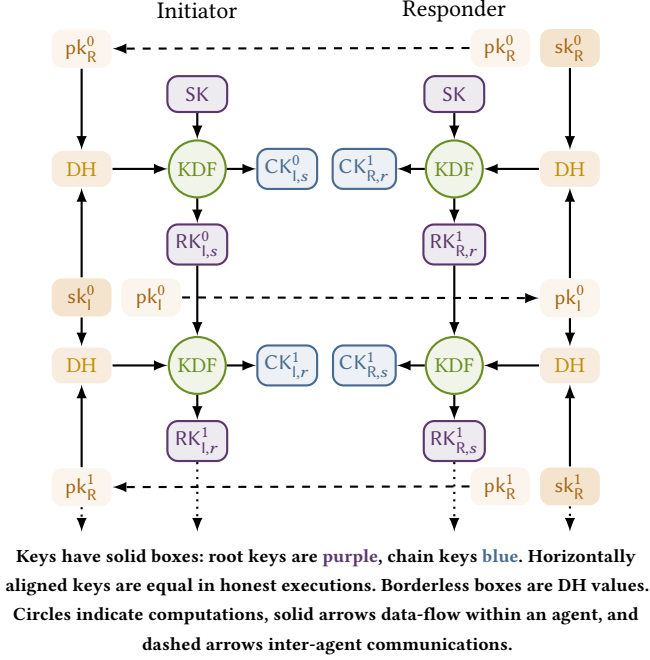


Figure 5: An asymmetric ratchet.

few that a user can easily manually re-derive using existing tactics, such as CDH.

Moreover, we extended SQUIRREL’s internals to connect the user-space $\triangleright$ and $\ntriangleright$ predicates to the existing automated deduction engine. This turned out to be crucial to reduce the proof-burden and to scale. First, a deduction $u \triangleright v$ can be automatically solved using the tactic **deduce** $\sim$ all, which relies on the deduction engine: this way, the user does not have to explicitly provide the computation justifying the deduction, which would be unacceptably tedious. Interestingly, the deduction engine also helps non-deduction. For example, consider a proof context with a hypothesis H stating that $u' \ntriangleright v'$, and where the current goal is $u \ntriangleright v$. We have extended the existing **apply** tactic so that **apply** H will use the deduction engine, attempting to automatically show that $u' \triangleright u$ and $v \triangleright v'$ and conclude by the weakening rules **WEAKL** and **WEAKR**. Finally, we made it so that **deduce** (without the $\sim$ all flag) simplifies goals even if it cannot prove them completely: e.g. to prove $u_1, \dots, u_n \ntriangleright v$, **deduce** will filter the elements of $(u_1, \dots, u_n)$ that can be deduced from the others, yielding a simpler goal.

4 The Security of our Asymmetric Ratchet

We model and prove the PCS of an asymmetric ratchet in SQUIRREL. This section presents the protocol, and the security property, explaining how they are modeled in our SQUIRREL development; and gives an in-depth explanation of how the proof is structured. All models and proofs are available at [38].

4.1 The Asymmetric Ratchet Protocol

We consider an asymmetric ratchet, whose execution is depicted in Fig. 5. A summary of all notations can be found in Table 2 in

appendix. This is a two-party protocol, in which an initiator and a responder both compute a sequence of *root keys*:

Initiator : $RK_{l,s}^0, RK_{l,r}^1, RK_{l,s}^2, RK_{l,r}^3, \dots$

Responder : $RK_{r,r}^1, RK_{r,s}^2, RK_{r,r}^3, RK_{r,s}^4, \dots$

In an ideal execution where the adversary does not intervene, both sequences are equal, i.e. $RK_{l,s}^0 = RK_{r,r}^1, RK_{l,r}^1 = RK_{r,s}^2$, etc. The role of each key depends on the agent, e.g. $RK_{l,s}^0$ is a *sending* key for the initiator, and should be equal to $RK_{r,r}^1$, which is a *receiving* key for the responder. For each agent, the chain alternates between sending and receiving keys. The two agents take turns in running the protocol. Every time an agent executes a protocol step, the chain progresses by two steps, by computing the receiving key that should match the last sending key from the other agent, and deriving a new sending key – the first execution of the initiator is different, as there is no receiving key, only a sending key $RK_{l,s}^0$. Except for this first step from the initiator, the behaviors of both agents are symmetric.

Each root key has an associated *chain key*, e.g. $CK_{R,r}^i$ is the i -th responder receiving chain key, and corresponds to $RK_{R,r}^i$ (these chain keys are not to be confused with the similarly named ones in Section 2). In Signal’s Double Ratchet, chain keys serve as starting points for symmetric ratchets, which we do not model here.

We assume that the initiator and responder chains are initialized with a shared secret SK , i.e. $RK_{l,s}^0 = RK_{r,r}^1 = SK$. In a real-world deployment, SK would be obtained by running e.g. the X3DH or PQXDH key exchanges, which we do not model here.

In an honest execution, the responder updates its key as follows. The i -th time it runs, it receives from the network a DH public key $pk_I^{i-1} = g^{sk_I^{i-1}}$, where g is a generator of a DH group – in an actual execution, the value received X may or may not be equal to pk_I^{i-1} . Then, it samples a fresh DH secret sk_R^i , and retrieves its previous DH secret sk_R^{i-1} from memory. It then performs two ratcheting steps. First, the i -th *receiving* root key $RK_{R,r}^i$ and chain key $CK_{R,r}^i$ are computed:

$$RK_{R,r}^i \parallel CK_{R,r}^i = H(RK_{R,s}^{i-1}, g^{sk_R^{i-1} \cdot sk_I^{i-1}})$$

Then, the i -th *sending* root and chain keys $RK_{R,s}^i, CK_{R,s}^i$ are derived:

$$RK_{R,s}^i \parallel CK_{R,s}^i = H(RK_{R,r}^i, g^{sk_R^i \cdot sk_I^{i-1}})$$

4.2 Modeling the Ratchet and PCS in Squirrel

We model this ratchet using SQUIRREL’s π -calculus process input language. It is then automatically compiled by the tool to an internal representation using recursive functions, as described in Fig. 2 of Section 2. Our model is displayed in Fig. 6. For concision, we omit the description of the initiator, as it is similar to the responder. In the following paragraphs, (numbers) refer to lines of Fig. 6.

Protocol. As SQUIRREL relies on an eager-sampling semantics, we start by sampling the randomness used in the ratchet. For example, the name SK (1) is the secret initializing the key exchange; and sk_R (2) contains the responder’s DH secrets. For consistency with earlier notations, the DH secret is denoted by sk_R^i rather than $sk_R(i)$. The responder’s global states $RK_{R,r}$ and $RK_{R,s}$ are then declared (4,5).

```

1 name SK : message. (* shared initial secret *)
2 name skR : index → E. (* array of responder DH secret exponents *)
3 name k, k' : message. (* PRF keys to model ROs *)
4 mutable RKR,r : message = empty. (* receiving root key *)
5 mutable RKR,s : message = SK. (* sending root key *)
6 process Responder(i : index) = (* i is the loop counter, starting at 1 *)
7   in(X); (* input new DH public key *)
8   RKR,r := h(⟨ RKR,s, XskRi-1 ⟩, k); (* update receiving root key *)
9   let CKR,r = h(RKR,r, k') in (* and chain key *)
10  RKR,s := h(⟨ RKR,r, XskRi ⟩, k); (* update sending root key *)
11  let CKR,s = h(RKR,s, k) in (* and chain key *)
12  stRCor := if safer(i) then zero else ⟨ RKR,s, skRi ⟩ (* leak state if not safe *)
13  out(⟨ gskRi, CKR,r, CKR,s ⟩)
14 ... (* omitted code defining the Initiator *)
15 process RO(j : index) = in(m); out(h(m, k)) (* random oracle for h *)
16 process RO'(j : index) = in(m); out(h'(m, k')) (* random oracle for h' *)
17 process Corrupt(j : index) = out(⟨ stRCor, stICor ⟩).
18 system S = ... (* initial knowledge + first initiator half-ratchet (omitted) *)
19 !_i Responder(i) | !_i Initiator(i) | !_j Corrupt(j) | !_j RO(j) | !_j RO'(j).

```

X := **t** assigns **t** to the global memory cell **X**. **in(x)** stores a network input in **x**, and **out(t)** outputs **t** to the network. (**P** | **Q**) runs **P** and **Q** in parallel. Finally, **!_i P(i)** replicates **P**, creating a copy for each value of **i**.

Figure 6: SQUIRREL modeling of the ratchet (simplified).

They store resp. the current responder receiving and sending root keys. Initially, there is no receiving key ($RK_{R,r}$ defaults to empty), and the sending root key is set to SK.

A single call to the KDF H simultaneously computes the new root key RK and chain key CK. We represent this KDF by:

$$H(x) = h(x, k) || h'(h(x, k), k')$$

where, as in Section 2, h and h' are random oracles, which we model using a PRF with a secret key, resp. k and k' , represented by names (3). Here, h produces RK (8,10) and h' produces CK (9,11). As mentioned, although we focus on the asymmetric ratchet, the first chain key of each symmetric chain is computed, and leaked (13). This amounts to proving security against a strong attacker, who could compromise every chain key. Finally, the adversary has access to $h(\cdot, k)$ and $h'(\cdot, k')$ using the RO and RO' processes (15,16).

Compromise. The adversary can compromise any agent's persistent memory. After step i of the responder, its memory contains the sending root key $RK_{R,s}^i$ and the DH secret sk_R^i , as both values must be stored for the next ratchet step. All other values, e.g. the receiving root key $RK_{R,r}$, are assumed to be deleted as soon as the i -th step ends. We model the compromise of the responder's state using a SQUIRREL memory cell $stRCor$, which is leaked to the adversary when it calls **Corrupt** (17) – similarly, $stICor$, omitted here, is used for the compromise of the initiator. Often, when modeling corruption, one observes *after the fact* which data was corrupted, and uses this information in the security statement, e.g. by requiring that uncorrupted sessions are secure. We take a different approach, and restrict *before the fact* which sessions may be compromised. We consider an arbitrary boolean function $safer(i)$ indicating whether the responder at step i is safe or may be compromised, and store the data to be leaked in $stRCor$ only when $safer(i)$ is false (12).

```

let rec syncRI(i:index) =
  i = 0 ∨
  (R(i) < I(i) ∧
   syncIR(i-1) ∧
   input(R(i))skRi = input(I(i))skIi-1)
and syncIR(i:index) = ...

let rec healthyI(i:index) =
  saferI(i) ∧
  ( (if syncRI(i) then healthyR(i)
    else healthyI(i-1)) ∨
    (saferR(i) ∧ input(I(i)) = gskRi) )
and healthyR(i:index) = ...

```

Figure 7: The synchronicity and health predicates.

This restriction is crucial for our study of PCS, as it is needed in order to use SQUIRREL's idiomatic proof style. Of course, restricting beforehand which sessions can be compromised is not equivalent to observing compromises after the fact. The former yields a *non-adaptive* formulation of PCS, in which the $safer(i)$ function is a parameter of the model: we prove PCS for any arbitrary but fixed set of corrupted sessions, but we do not let the adversary dynamically choose which sessions to corrupt.

Healthy predicate and PCS. The full protocol consists of the replication of all processes (18,19) running in parallel. It gives rise to the set of timestamps (some details omitted):

$$\text{timestamp} ::= \text{Init} \mid R(i) \mid I(i) \mid \text{Corrupt}(j) \mid RO(j) \mid RO'(j)$$

where $R(i)$ and $I(i)$ are resp. the i -th responder and initiator steps, and other timestamps are as expected from their names.

We focus on the secrecy of the sending keys, as this is what would be needed to prove message confidentiality when extending the ratchet into a complete secure messaging protocol. Naturally, not all $RK_{I,s}^i$ are expected to be secret – when a corruption has occurred, and the protocol has not healed yet, the keys are not meant to be secret. Hence, we use a predicate $healthy_I(i)$, indicating whether $RK_{I,s}^i$ will be shown secret, and, similarly, $healthy_R$ for the responder key $RK_{R,s}$. Once the healthy predicates are defined – which will follow shortly – we capture PCS by requiring that healthy responder and initiator sending keys are secret. Formally, for the initiator:

global lemma $PCS^I(\tau: \text{timestamp}[\text{const}], i: \text{index}[\text{const}]):$
 $[\text{happens}(\tau, I(i))] \Rightarrow [\text{healthy}_I(i)] \Rightarrow (\text{frame}(\tau) \not\models RK_{I,s}^i).$

The goal, when defining $healthy_I$, is for as many keys as possible to be considered healthy, so that the resulting property is as strong as possible. We show in our SQUIRREL development that PCS also implies PFS, by proving that the $healthy_I(i)$ we will define holds if there is no corruption before $I(i)$.

Before defining $healthy$, it is useful to characterize when the initiator and responder are synchronized, i.e. when they derive the same key chains up to step i . We capture this using two recursive predicates, seen on the left of Fig. 7: $sync_{RI}(i)$ indicates whether the i -th responder-to-initiator keys are synchronized, i.e. whether the i -th responder sending key is equal to the i -th initiator receiving key; $sync_{IR}(i)$ is similar for the initiator-to-responder keys.

Armed with the sync predicates, we can now define $healthy$ keys. The definition of $healthy_I$ is given on the right of Fig. 7, and $healthy_R$ is similar. The two predicates are mutually recursive. First, $healthy_I(i)$ requires that $safer_I(i)$ holds, as $RK_{I,s}$ cannot be proved secret if the adversary can directly compromise it. Then, the secrecy of $RK_{I,s}^i = h(\langle RK_{I,r}^i, \text{input}(I(i))^{sk_I^i} \rangle, k)$ can stem from the secrecy of either the previous root key (left) or the DH handshake (right).

Component	Filename	LoC	Time
Model definitions	model.sp	226	3s
Security definitions	main.sp	150	1s
Definition of $\mathcal{O}$	oracle.sp	838	7s
Simulation proof	simulation.sp	305	14s
Non-collisions proofs	non_collision.sp	1212	46s
Secrecy proof	secrecy_p*.sp	3365	236s
Additional utilities	indices.sp, trace.sp, DHlib.sp, utils.sp	1553	26s
Total		7649	333s

Table 1: Lines of Code (LoC) for the SQUIRREL proof of the ratchet, including comments and white spaces. Verification times are for a standard laptop (4 cores@2.8GHz CPU, 32GB RAM). Corresponding files are in the artifact [38].

Case 1: inherited secrecy. The secrecy of $RK_{l,s}^i$ may rest upon the secrecy of the previous root key $RK_{l,r}^i$. When both agents are synchronized, $RK_{l,r}^i = RK_{R,s}^i$, and we ensure the secrecy of the former by requiring that the latter is healthy. When agents are de-synchronized, the secrecy of $RK_{l,r}^i = h(\langle RK_{l,s}^{i-1}, \dots \rangle, k)$ can only be achieved from that of $RK_{l,s}^{i-1}$, since the rest of the hashed pair depends on an adversarial input. Thus, we ask that $RK_{l,s}^{i-1}$ is healthy, *i.e.* that $\text{healthy}_l(i-1)$ holds.

Case 2: healing. $l(i)$ can be a healing step, where the secrecy of the root key $RK_{l,s}^i$ rests upon the secrecy of the honest DH handshake $g^{\text{sk}_l^i \cdot \text{sk}_R^i}$. We already required that $\text{safe}_l(i)$, thus sk_l^i is not leaked. sk_R^i must not leak either, *i.e.* $\text{safe}_R(i)$ must hold, and the initiator must receive the expected DH value $g^{\text{sk}_R^i}$ from the network.

As we did for the safe predicate, we assume that the values of the healthy and sync predicates are fixed before the start of the protocol execution, *i.e.* we consider a non-adaptive adversary, who must decide beforehand, rather than dynamically, which sessions are attacked or allowed to heal, and when to de-sync the protocol.

Discussion (non-adaptiveness). We quickly discuss why we target a non-adaptive form of PCS. From a foundational point-of-view, there is no inherent need for non-adaptiveness: on paper, the logic can express the adaptive notion of PCS expected by cryptographers. The limitation stems from the idiomatic proof methodology in SQUIRREL, which does exploit non-adaptiveness — the proof strategy we use for this ratchet is no exception. More precisely, while SQUIRREL’s rules do not explicitly require non-adaptiveness, some rules impose it indirectly. Without delving too much in technicalities, one form of case analysis (the **G.L-LOC:V** rule from [8]) is limited to constant values. It thus cannot be applied to adversarially controlled values: as the adversary is probabilistic, values it produces cannot be assumed constant. Further down the road, this usually forces some non-adaptiveness in the model.

4.3 Security Proof

We provide in [Table 1](#) an overview of the SQUIRREL development, including the number of lines of code (LoC) for each file. We use the

```

let  $\mathcal{O}_{\text{leak}}^R = \lambda i. \text{if not } (\text{safe}_R(i)) \text{ then } \text{sk}_R^i$ 
let  $\mathcal{O}_{\text{public}}^R = (\lambda x. h(x, k), \lambda x. h'(x, k'), \lambda i. g^{\text{sk}_R^i}, \mathcal{O}_{\text{leak}}^R, \mathcal{O}_{\text{ddh}}^R, \dots)$ 
let  $\mathcal{O}_{\text{RK}}^{\text{R},s}(\tau) = \lambda i. \text{if } R(i) \leq \tau \wedge \text{not}(\text{healthy}_R(i)) \text{ then } RK_{R,s}^i$ 
let  $\mathcal{O}_{\text{RK}}^{\text{R},r}(\tau) = \lambda i. \text{if } R(i) \leq \tau \wedge \text{not}(\text{sync}_{\text{IR}}(i-1)) \wedge \text{not}(\text{healthy}_R(i-1)) \text{ then } RK_{R,r}^i$ 
let  $\mathcal{O}_{\text{CK}}^{\text{R},s}(\tau) = \lambda i. \text{if } R(i) \leq \tau \wedge \text{healthy}_R(i) \text{ then } CK_{R,s}^i$ 
let  $\mathcal{O}_{\text{CK}}^{\text{R},r}(\tau) = \lambda i. \text{if } R(i) \leq \tau \wedge \text{healthy}_R(i-1) \wedge \text{not}(\text{sync}_{\text{IR}}(i-1)) \text{ then } CK_{R,r}^i$ 
let  $\mathcal{O}(\tau) = (\mathcal{O}_{\text{public}}^R, \mathcal{O}_{\text{RK}}^{\text{R},s}(\tau), \mathcal{O}_{\text{RK}}^{\text{R},r}(\tau), \mathcal{O}_{\text{CK}}^{\text{R},s}(\tau), \mathcal{O}_{\text{CK}}^{\text{R},r}(\tau), \dots)$ 

```

Figure 8: The set of oracles $\mathcal{O}(\tau)$ (excerpt).

same high-level proof structure as in the motivating example in [Section 2.3](#): *first*, we prove a number of non-collision properties on ratchet keys; *second*, we prove that the protocol execution $\text{frame}(\tau)$ can be simulated by a well-chosen set of oracles $\mathcal{O}(\tau)$; *third*, we prove the secrecy of the target keys under $\mathcal{O}$. We do not further discuss here the proofs of the non-collision properties, which proceed similarly to [Section 2](#).

[Fig. 9](#) depicts one possible execution of the ratchet, which we use to illustrate our proof strategy throughout this section. It also makes clear the bug uncovered in the seminal pen-and-paper proof of the ratchet [25], discussed in [Example 1.1](#). We see in [Fig. 9](#) that the root key $RK_{R,r}^1$ is secure and non-deducible. It is however not RoR, as the attacker can obtain through compromise $RK_{R,s}^1$ and sk_R^1 , which yields a trivial distinguisher. The potential for such compromises to break RoR was missed in [25], where RoR is typically claimed for $RK_{R,r}^1$ in the proofs (see [Appendix E](#) for details).

4.3.1 Proof cut by simulation. Rather than directly proving that $\text{frame}(\tau) \not\models \text{RK}$, we identify a set of oracles $\mathcal{O}(\tau)$ such that

$$\mathcal{O}(\tau) \triangleright \text{frame}(\tau) \quad \text{and} \quad \mathcal{O}(\tau) \not\models \text{RK}.$$

That is, $\mathcal{O}$ gives at least as much knowledge to the attacker, and does not break secrecy. We present the main oracles of $\mathcal{O}$ in [Fig. 8](#) (see the full definition in [Fig. 13](#) in appendix). First, $\mathcal{O}$ contains a set of oracles $\mathcal{O}_{\text{public}}$ independent of τ , which includes the random oracles for h and h' , the public DH keys, the leaked secret DH keys, and DDH oracles for all secret DH keys. Then, $\mathcal{O}(\tau)$ contains a set of root and chain keys, sufficiently large to simulate $\text{frame}(\tau)$, while avoiding duplicated keys (we justify this design choice later). For example, $\mathcal{O}(\tau)$ includes all non-healthy sending root keys appearing before τ , and all receiving root keys in case of de-sync (when in sync, adding a receiving key would create a duplicate). $\mathcal{O}(\tau)$ also includes all sending chain keys corresponding to healthy root keys (a non-healthy CK can be computed from the corresponding RK and the hash oracle $h'(\cdot, k')$). We discuss below the two main benefits of introducing the intermediate set of oracles $\mathcal{O}$.

Removing duplicate hashes. Recall the PRF rules of [Section 3](#):

$$\frac{u, o_h^k \not\models v \quad \dots}{u, o_h^k, h(m, k) \not\models v} \quad \frac{u, o_h^k \not\models m \quad \dots}{u, o_h^k \not\models h(m, k)}$$

Reasoning backwards, the left rule removes a hash from the adversary’s knowledge, while the right rule reduces the secrecy of $h(m, k)$ to that of m . The additional conditions, omitted here, notably require that the hash $h(m, k)$ is fresh, *i.e.* that m is not hashed

in u, v . But some hashes in the execution $\text{frame}(\tau)$ of the ratchet are *not* fresh, as the initiator and responder compute the same hash chains while in sync: e.g. in Fig. 9, the chain keys $\text{CK}_{R,r}^1$ and $\text{CK}_{I,s}^1$ are equal, and the key $\text{CK}_{R,r}^1$ thus appears twice in $\text{frame}(\tau)$. Using O , we remove such duplicate hashes to prepare the terrain for the application of the PRF rules during the secrecy proof.

Robustness. Simulating $\text{frame}(\tau)$ by O abstracts away irrelevant implementation details. Superficial modifications of the protocol model, e.g. changing a data structure or a serialization function, will only impact a small part of the simulation proof, and will not require modifying O . There are two advantages to that. First, as seen in Table 1, the simulation proof is a small subset of the overall development, as it benefits the most from the automated deduction procedure presented in Section 3.5. It is thus easy to adapt when updating the proof. Second, since O is unchanged, there is no need to update the secrecy proof $O(\tau) \not\vdash \dots$, which is the most intricate and time-consuming proof step.

4.3.2 Inductive proof of secrecy. To prove PCS, we must establish properties of the form $O(\tau) \not\vdash \text{RK}$. Our high-level proof strategy consists in recursively eliminating in $O(\tau)$ or RK the keys at the end of the longest KDF chains, i.e. the most recently computed keys. This yields an inductive proof in which we show – in some well-chosen order – the secrecy of all healthy keys, and of some useful DH values. Let $\text{ir}(\tau_r)$ and $\text{il}(\tau_r)$ be, respectively, the largest $\text{R}(i)$ and $\text{I}(i)$ timestamps earlier than τ . We target the property¹:

- 1 **global lemma** $\text{secrecy}_{\text{main}}(\tau_l: \text{timestamp}[\text{const}])$:
- 2 $\checkmark (n, n': \text{index}[\text{const}]), [\text{safe}_i(n) \wedge \text{safer}_R(n')] \Rightarrow O(\tau_l) \not\vdash g^{\text{sk}_I^{\tau_l} \text{sk}_R^{\tau_l'}}$
- 3 $\checkmark (\tau_r: \text{timestamp}[\text{const}]),$
- 4 $[\text{healthy}_R(\text{ir}(\tau_r))] \Rightarrow O(\tau_l) \not\vdash \text{RK}_{R,s}^{\text{ir}(\tau_r)} \hat{\wedge}$
- 5 $[\text{healthy}_I(\text{il}(\tau_r))] \Rightarrow O(\tau_l) \not\vdash \text{RK}_{I,s}^{\text{il}(\tau_r)} \hat{\wedge}$
- 6 $[\dots] \Rightarrow O(\tau_l) \not\vdash \text{RK}_{R,r}^{\text{ir}(\tau_r)} \hat{\wedge} [\dots] \Rightarrow O(\tau_l) \not\vdash \text{RK}_{I,r}^{\text{il}(\tau_r)}$

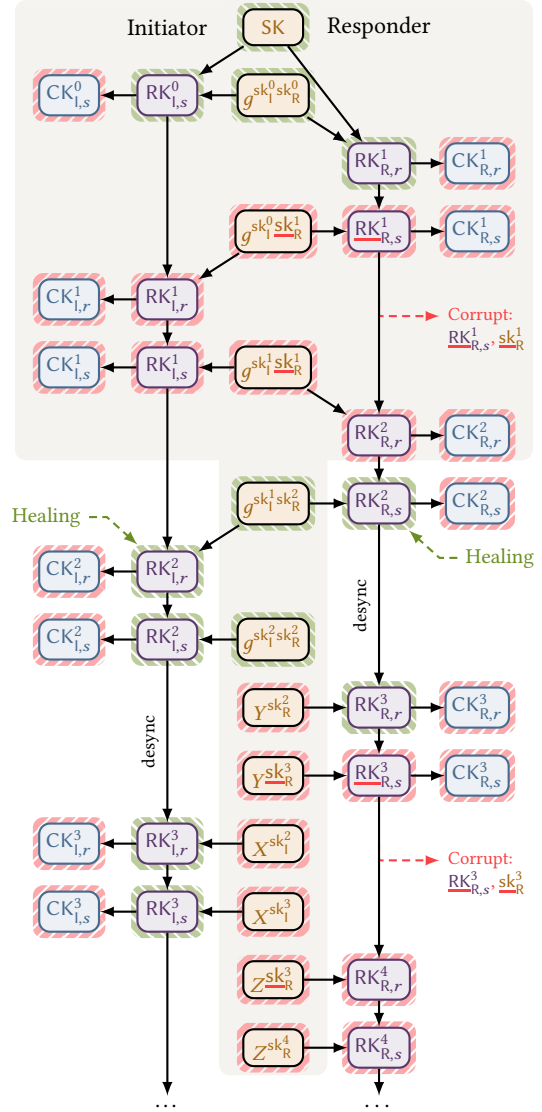
To prove this lemma, it suffices to consider the case $\tau_r \leq \tau_l$. Indeed, when $\tau_l < \tau_r$, we observe that $O(\tau_r) \supset O(\tau_l)$. Thus, using the weakening rule **WEAKL**, it suffices to prove $O(\tau_r) \not\vdash \text{K}(\tau_r)$ to obtain $O(\tau_l) \not\vdash \text{K}(\tau_r)$, bringing us back to an instance of the previous case.

The proof then proceeds by induction on τ_l . At each step, we either apply a right PRF rule to reduce the secrecy of a key to that of an earlier key, or a left PRF rule to remove from $O(\tau)$ a key that is no longer available in $O(\text{pred}(\tau))$. In total, the proof contains 23 uses of our PRF rules, divided among **PRF.R**, **PRF.L-EQ**, and **PRF.L-SEC**. We finish with a few interesting cases of the induction.

Base case. When τ_l is the initial timestamp, $O(\text{Init})$ is limited to the public values O_{public} , and only the non-deduction at line 2 of lemma $\text{secrecy}_{\text{main}}$ is not vacuous. Thus, we must prove the secrecy of all safe DH products from O_{public} – in Fig. 9, these are the values with a green aura in the middle column. We show this directly by reduction to Gap Diffie-Hellman (GDH).

Healing case. Let $\tau_l = \text{R}(2)$ and consider the proof of $O(\tau_l) \not\vdash \text{RK}_{R,s}^2$ in Fig. 9. This is a healing step in which the responder received the expected uncompromised DH public value $g^{\text{sk}_I^1}$, freshly sampled sk_R^2 , and computed a fresh secret $g^{\text{sk}_I^1 \cdot \text{sk}_R^2}$ that it injected in the KDF

¹Omitting some details for simplicity; the corresponding predicate can be found in [38], under the name `Secret_prop` in file `Models/asym-ratchet/secrecy_p1.sp`.



Compromised values are underlined. Corruption events are red dashed arrows. Healthy / unhealthy nodes have green / red auras.

Figure 9: Asymmetric ratchet proof visualization

derivation. The gray background area in Fig. 9 indicates which values are known to be secret at that point: for any value K with a green aura in that zone, we know by induction hypothesis that $O(\tau_l) \not\vdash K$. Notably, $g^{\text{sk}_I^1 \cdot \text{sk}_R^2}$ is secret, which allows us to prove the secrecy of $\text{RK}_{R,s}^2$ using the **PRF.R** rule. To apply this rule, we must additionally prove that $\text{RK}_{R,s}^2$ is a fresh hash, which we show using the non-collision lemmas and by proving that $\text{RK}_{R,s}^2$ was never sent to RO. While this may at times require the induction hypothesis, it follows here from the fact that sk_R^2 is a fresh random sampling.

Inherited healthy case. Consider the value $\text{RK}_{I,s}^3$ in Fig. 9. This key may have used a DH share known to the attacker in its derivation.

However, $RK_{l,s}^3$ is also derived from the healthy key $RK_{l,r}^3$, which is already established secret at that point in the induction. Thus, the secrecy of $RK_{l,s}^3$ can be proved with a call to `PRF.R`.

Application of a left PRF rule. $RK_{R,s}^4$ illustrates the application of a left PRF rule to eliminate a key from $O(\tau)$. Depending on which value the adversary chooses for Z , it might or might not be able to compute $RK_{R,s}^4$ – roughly, it can compute this key only if it knows the discrete log of Z . Because $RK_{R,s}^4$ may not be secret, we cannot use `PRF.L-SEC` (as we do in other cases), and rely instead on `PRF.L-EQ`. This particular step shows how the computation of a root key by an agent can provide the attacker with a DDH oracle for that key, and explains the inclusion of such an oracle in O_{public} .

5 Conclusion

We have now verified the PCS of an asymmetric ratchet. A natural next step will be to consider the full DR. Analyzing the post-quantum security of the SPQR protocol [28], by extending non-deduction to post-quantum `SQUIRREL` [3], is another exciting avenue for future work. This would likely require reasoning in the QROM [22], which is known to be difficult in CAC tools [12]. We suspect that this problem could be tackled by exploiting the similarities between our `PC-ORACLE` rule and One-Way to Hiding lemmas [48]. Finally, we leave as future work to apply [7] to obtain security guarantees in our case-study in the concrete setting.

Acknowledgments

This work benefited from funding managed by the French National Research Agency under the France 2030 programme with the reference ANR-22-PECY-0006 and ANR-22-PETQ-0008.

References

- [1] Joël Alwen, Sandro Coretti, and Yevgeniy Dodis. 2019. The double ratchet: security notions, proofs, and modularization for the signal protocol. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 129–158.
- [2] Apple. [n. d.]. iMessage with PQ3: The new state of the art in quantum-secure messaging at scale. <https://security.apple.com/blog/imessage-pq3/>.
- [3] David Baelde, Antoine Dallon, Stéphanie Delaune, Charlie Jacomme, and Adrien Koutsos. 2026. Robust Logical Foundations for Mechanizing Post-Quantum Cryptography in Squirrel. In *to appear in CCS'26*. <https://inria.hal.science/hal-0595035>.
- [4] David Baelde, Stéphanie Delaune, Charlie Jacomme, Adrien Koutsos, and Solène Moreau. 2021. An Interactive Prover for Protocol Verification in the Computational Model. In *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*. IEEE, 537–554.
- [5] David Baelde, Stéphanie Delaune, Adrien Koutsos, and Solène Moreau. 2022. Cracking the Stateful Nut: Computational Proofs of Stateful Security Protocols using the Squirrel Proof Assistant. In *CSF. IEEE*, 289–304.
- [6] David Baelde, Stéphanie Delaune, and Stanislas Riou. 2025. SMT-Based Automation for Overwhelming Truth. In *CSF. IEEE*, 505–520.
- [7] David Baelde, Caroline Fontaine, Adrien Koutsos, Guillaume Scerri, and Théo Vignon. 2024. A Probabilistic Logic for Concrete Security. In *CSF. IEEE*, 324–339.
- [8] David Baelde, Adrien Koutsos, and Joseph Lallemand. 2023. A Higher-Order Indistinguishability Logic for Cryptographic Reasoning. In *LICS. IEEE*, 1–13.
- [9] David Baelde, Adrien Koutsos, and Justine Sauvage. 2024. Foundations for Cryptographic Reductions in CCSA Logics. In *CCS. ACM*, 2814–2828.
- [10] David Baelde, Adrien Koutsos, and Justine Sauvage. 2026. Leveraging Cryptographic Simulator Synthesis for Formally Verifying the FOO E-Voting Protocol. In *to appear in Usenix 2026 - 35th Usenix security symposium*. Baltimore, United States. <https://inria.hal.science/hal-05453231>
- [11] Manuel Barbosa, Gilles Barthe, Karthik Bhargavan, Bruno Blanchet, Cas Cremers, Kevin Liao, and Bryan Parno. 2021. SoK: Computer-Aided Cryptography. In *SP. IEEE*, 777–795.
- [12] Manuel Barbosa, Gilles Barthe, Xiong Fan, Benjamin Grégoire, Shih-Han Hung, Jonathan Katz, Pierre-Yves Strub, Xiaodi Wu, and Li Zhou. 2021. EasyPQC: Verifying Post-Quantum Cryptography. In *CCS. ACM*, 2564–2586.
- [13] Gilles Barthe, Benjamin Grégoire, Sylvain Heraud, and Santiago Zanella-Béguelin. 2011. Computer-Aided Security Proofs for the Working Cryptographer. In *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings (Lecture Notes in Computer Science, Vol. 6841)*, Phillip Rogaway (Ed.). Springer, 71–90. [doi:10.1007/978-3-642-22792-9_5](https://doi.org/10.1007/978-3-642-22792-9_5)
- [14] Gilles Barthe, Benjamin Grégoire, and Santiago Zanella-Béguelin. 2009. Formal certification of code-based cryptographic proofs. In *POPL. ACM*, 90–101.
- [15] David A. Basin, Andreas Lochbihler, and S. Reza Sefidgar. 2020. CryptHOL: Game-Based Proofs in Higher-Order Logic. *J. Cryptol.* 33, 2 (2020), 494–566.
- [16] Karthikeyan Bhargavan, Abhishek Bichhawat, Quoc Huy Do, Pedram Hosseini, Ralf Küsters, Guido Schmitz, and Tim Wurtel. 2021. DY*: A Modular Symbolic Verification Framework for Executable Cryptographic Protocol Code. In *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 523–542.
- [17] Alexander Bienstock, Jaiden Fairoze, Sanjam Garg, Pratyay Mukherjee, and Srinivasan Raghuraman. 2022. A more complete analysis of the signal double ratchet algorithm. In *Annual International Cryptology Conference*. Springer, 784–813.
- [18] Bruno Blanchet. 2008. A Computationally Sound Mechanized Prover for Security Protocols. *IEEE Trans. Dependable Secur. Comput.* 5, 4 (2008), 193–207. [doi:10.1109/TDSC.2007.1005](https://doi.org/10.1109/TDSC.2007.1005)
- [19] Bruno Blanchet. 2017. Symbolic and Computational Mechanized Verification of the ARINC823 Avionic Protocols. In *CSF. IEEE Computer Society*, 68–82.
- [20] Bruno Blanchet et al. 2016. Modeling and verifying security protocols with the applied pi calculus and ProVerif. *Foundations and Trends® in Privacy and Security* 1, 1-2 (2016), 1–135.
- [21] Olivier Blazy, Ioana Boureanu, Pascal Lafourcade, Cristina Onete, and Léo Robert. 2023. How fast do you heal? A taxonomy for post-compromise security in secure-channel establishment. In *32nd USENIX Security Symposium (USENIX Security 23)*. 5917–5934.
- [22] Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. 2011. Random Oracles in a Quantum World. In *ASIACRYPT (Lecture Notes in Computer Science, Vol. 7073)*. Springer, 41–69.
- [23] Ran Canetti, Palak Jain, Marika Swanberg, and Mayank Varia. 2022. Universally composable end-to-end secure messaging. In *Annual International Cryptology Conference*. Springer, 3–33.
- [24] Vincent Cheval, Charlie Jacomme, and Jessica Richards. 2026. Automated formal analysis of Signal’s Double Ratchet: attacks, fixes and security proofs. In *47th IEEE Symposium on Security and Privacy, SP 2026, San Francisco, CA, USA, 18-21 May 2026*. <https://eprint.iacr.org/2026/727>
- [25] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. 2020. A formal security analysis of the signal messaging protocol. *Journal of Cryptology* 33, 4 (2020), 1914–1983.
- [26] Katriel Cohn-Gordon, Cas Cremers, and Luke Garratt. 2016. On Post-compromise Security. In *CSF. IEEE Computer Society*, 164–178.
- [27] Daniel Collins, Doreen Riepel, and Si An Oliver Tran. 2024. On the tight security of the double ratchet. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*. 4747–4761.
- [28] Graeme Connell and Rolf Schmidt. October 2025. Signal Protocol and Post-Quantum Ratchets. <https://signal.org/blog/spqr/>.
- [29] Cas Cremers, Charlie Jacomme, and Aurora Naska. 2023. Formal Analysis of Session-Handling in Secure Messaging: Lifting Security from Sessions to Conversations. In *USENIX Security Symposium*. USENIX Association, 1235–1252.
- [30] Stéphanie Delaune, Clément Hérourard, and Joseph Lallemand. 2025. Secrecy by typing in the computational model. In *2025 IEEE 38th Computer Security Foundations Symposium (CSF)*. IEEE, 17–32.
- [31] The Squirrel development team. 2026. *Squirrel user documentation*. <https://squirrel-prover.github.io/documentation/>
- [32] The Vodozemac development team. 2026. The Vodozemac implementation of Olm. <https://github.com/matrix-org/vodozemac>.
- [33] Whitfield Diffie and Martin E. Hellman. 1976. *New directions in cryptography*. *IEEE Trans. Inf. Theory* 22, 6 (1976), 644–654.
- [34] Danny Dolev and Andrew Chi-Chih Yao. 1981. On the Security of Public Key Protocols (Extended Abstract). In *FOCS. IEEE Computer Society*, 350–357.
- [35] Atsushi Fujioka, Tatsuaki Okamoto, and Kazuo Ohta. 1992. A Practical Secret Voting Scheme for Large Scale Elections. In *AUSCRYPT (Lecture Notes in Computer Science, Vol. 718)*. Springer, 244–251.
- [36] Joshua Gancher, Sydney Gibson, Pratap Singh, Samvid Dharanikota, and Bryan Parno. 2023. Owl: Compositional verification of security protocols via an information-flow type system. In *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1130–1147.
- [37] Philipp G. Haselwarter, Exequiel Rivas, Antoine Van Muylder, Théo Winterhalter, Carmine Abate, Nikolaj Sidorenko, Catalin Hritcu, Kenji Maillard, and Bas Spitters. 2023. SSProve: A Foundational Framework for Modular Cryptographic Proofs in Coq. *ACM Trans. Program. Lang. Syst.* 45, 3 (2023), 15:1–15:61.

POLYL $\frac{u \text{ has type } \tau \quad v \not\vdash w \quad \text{poly-enum} \in \text{label}(\tau)}{u, v \not\vdash w}$	POLYR $\frac{u \text{ has type } \tau \quad \text{poly-enum} \in \text{label}(\tau)}{\neg(u \not\vdash v)}$
--	--

Figure 10: Additional core rules for $\triangleright$ and $\not\vdash$.

- [38] Clément Hérourard, Charlie Jacomme, Adrien Koutsos, and Joseph Lallemand. 2026. *Secrecy in Squirrel and the Post-Compromise Security of a Ratchet*. doi:10.5281/zenodo.22204533
- [39] Nadim Kobeissi, Karthikeyan Bhargavan, and Bruno Blanchet. 2017. Automated verification for secure messaging protocols and their implementations: A symbolic and computational approach. In *2017 IEEE European symposium on security and privacy (EuroS&P)*. IEEE, 435–450.
- [40] Felix Linker, Ralf Sasse, and David Basin. 2025. A Formal Analysis of Apple’s iMessage PQ3 Protocol. In *USENIX Security Symposium*. USENIX Association.
- [41] Felix Emanuel Linker, Christoph Sprenger, Cas Cremers, and David Basin. 2025. Looping for Good: Cyclic Proofs for Security Protocols. In *32nd ACM Conference on Computer and Communications Security (CCS)*. Association for Computing Machinery.
- [42] Benjamin Lipp, Bruno Blanchet, and Karthikeyan Bhargavan. 2019. A Mechanised Cryptographic Proof of the WireGuard Virtual Private Network Protocol. In *EuroS&P*. IEEE, 231–246.
- [43] Trevor Perrin and Moxie Marlinspike. 2017. *The Sesame Algorithm: Session Management for Asynchronous Message Encryption*. <https://signal.org/docs/specifications/sesame/>
- [44] Adam Petcher and Greg Morrisett. 2015. The foundational cryptography framework. In *International Conference on Principles of Security and Trust*. Springer, 53–72.
- [45] Benedikt Schmidt, Simon Meier, Cas Cremers, and David A. Basin. 2012. Automated Analysis of Diffie-Hellman Protocols and Advanced Security Properties. In *25th IEEE Computer Security Foundations Symposium, CSF 2012, Cambridge, MA, USA, June 25–27, 2012*, Stephen Chong (Ed.). IEEE Computer Society, 78–94. doi:10.1109/CSF.2012.25
- [46] Techcrunch. 2025. *WhatsApp now has more than 3 billion users a month*. <https://techcrunch.com/2025/05/01/whatsapp-now-has-more-than-3-billion-users/>
- [47] Devon Tuma and Nicholas Hopper. 2024. VCVio: A Formally Verified Forking Lemma and Fiat-Shamir Transform, via a Flexible and Expressive Oracle Representation. *Cryptology ePrint Archive* (2024).
- [48] Dominique Unruh. 2015. Revocable quantum timed-release encryption. *Journal of the ACM (JACM)* 62, 6 (2015), 1–76.

A Open Science

Our artifact is available at [38]:

<https://doi.org/10.5281/zenodo.22204533>

This artifact includes:

- README.md: the top-level readme file.
- squirrel-prover/: our extension of SQUIRREL with support for non-deduction.
- case-studies/: our case-studies.

Further, the extension has been integrated into the master branch of SQUIRREL, which is available at:

<https://github.com/squirrel-prover/squirrel-prover/>

Installation. Installation instructions for our extension of SQUIRREL are available in squirrel-prover/README.md.

Proof visualization. However, it is not necessary to install SQUIRREL to visualize our case-studies, as Models/html_export contains an html export that can be consulted directly from a Web browser.

Secrecy library. The user-space SQUIRREL code for non-deduction can be found in:

squirrel-prover/theories/NonDeduction.sp

It notably proves some of our rules (Theorem C.1).

Case-studies. Our case-studies are organized as follows:

- Models/asym-ratchet/: the asymmetric ratchet case-study of Section 4, see the corresponding README.md for additional details.
- Models/motivating-example.sp: the model and proof for the motivating example of Section 2.
- Models/non_deduction_tuto.sp: a small tutorial introducing the basic techniques for reasoning on non-deduction in our extension of SQUIRREL.

B Motivating Example: Missing Details

We describe the high-level structure of the proof of the non-collision lemmas we presented in Section 2.

lemma NoCollision_{Init} (i:index): happens(A(i)) $\Rightarrow$ ck(A(i)) $\neq$ s.

lemma NoCollision_A (i,j:index): A(i) < A(j) $\Rightarrow$ safe(A(j)) $\Rightarrow$ ck(A(i)) $\neq$ ck(A(j)).

NoCollision_{Init} states that all non-initial chain keys are different from the initial seed s. It can be directly proven by reduction to the pre-image resistance of $H(\cdot, k)$. We prove NoCollision_A by induction over the execution trace $<$. First, we use the collision-resistance of $H(\cdot, k)$ to reduce the existence of a collision

$$\text{ck}(A(i)) = \text{ck}(A(j))$$

to the existence of a collision:

$$\text{ck}(\text{pred}(A(i))) = \text{ck}(\text{pred}(A(j))) \quad (2)$$

Then, we conclude by a case-analysis, depending on whether the left chain key is the initial key s or a key $\text{ck}(A(i_0))$ where $A(i_0) < A(i)$. In the former case, we conclude thanks to NoCollision_{Init}. In the latter case, there exists j_0 such that Eq. (2) is equivalent to

$$\text{ck}(A(i_0)) = \text{ck}(A(j_0))$$

and we conclude by induction hypothesis.

C Additional Rules and Proofs

We consider the usual global judgement $\mathcal{E}; \Theta \vdash F$, where $\mathcal{E}$ is an environment, Θ is a set of global formulas and forms the judgement’s hypotheses, and the conclusion F a global formula. This global judgement is valid when $(\tilde{\Lambda}\Theta) \Rightarrow F$ is satisfied in all models of $\mathcal{E}$. Our proof system is composed of rules of the form

$$\frac{J_1 \quad \dots \quad J_k}{J},$$

where J and the J_i are global judgements, which we prove correct, in the sense that if $J_1, \dots, J_k$ are valid then so is J .

We omit the environment $\mathcal{E}$ and hypotheses Θ from a rule if they are identical in all its judgements. The rules presented in the body of this paper (in Fig. 3 and Fig. 4) should also be read this way.

C.1 Core Rules

We present in Fig. 10 two additional core rules that we omitted from the body. Rules **POLYL** and **POLYR** express a cardinality argument: if a term v is of a type that can be enumerated in polynomial-time, then it can always be guessed with non-negligible probability by an adversary, and thus it both cannot be secret, and is not useful to deduce other terms.

All our core rules are sound.

THEOREM C.1. *The rules of Fig. 3 are sound.*

PROOF. All the rules but **POLYL**, **POLYR**, **RW-ORACLE** and **PC-ORACLE** are proved directly in **SQUIRREL**, and their proof can be found in our artifact [38].

*The **POLYL** and **POLYR** rules.* First, the poly-enum label in $\text{label}(\tau)$ means that τ is interpreted, in all models $\mathbb{M}$ considered, as an η -indexed family of sets $\llbracket \tau \rrbracket_{\mathbb{M}}^{\eta}$ whose elements can be enumerated in a polynomial time in η . Therefore, in a given model, an adversarial function could thus enumerate all possible values in $\llbracket \tau \rrbracket_{\mathbb{M}}^{\eta}$, and sample one of them at random. The label poly-enum implies that the cardinal of $\llbracket \tau \rrbracket_{\mathbb{M}}^{\eta}$ is polynomial in η . So, the probability of obtaining $\llbracket u \rrbracket_{\mathbb{M}}^{\eta}$ with a uniform random sampling is non-negligible. Thus, the value of u cannot be secret, which justifies rule **POLYR**, and is not needed to deduce any other value, as it could be simulated, which justifies rule **POLYL**.

*The **PC-ORACLE** and **RW-ORACLE** rules.* **PC-ORACLE** is derived directly from **RW-ORACLE**, hence we focus on the latter. **RW-ORACLE** cannot be proved directly in **SQUIRREL**, since, as we will see, it involves reasoning inductively on the number of calls made by an adversary to an oracle, which the tool currently cannot do.

We first present a slight reformulation and generalization of **RW-ORACLE**, noticing that it is in fact symmetric between the two oracles: the second premise asks that the two oracles behave the same on inputs deducible from u, o_1 , but considering instead inputs deducible from u, o_2 would be sound as well. We thus prove the soundness of the following (double) rule at once, for both $\beta = 1$ and $\beta = 2$.

$$\frac{u, o_2 \not\vdash v \quad \forall m. (u, o_{\beta} \triangleright m) \Rightarrow [o_1 m = o_2 m]}{u, o_1 \not\vdash v}$$

Assuming that the rule's premises hold, we will prove its conclusion. Unfolding the definition of $\not\vdash$, consider an adversarial function f , that will be applied to u, o_1 to attempt to compute v . Its interpretation $\llbracket f \rrbracket^{\eta, \rho}$ is an adversary $\mathcal{A}(1^{\eta}, \rho_a, \cdot, \cdot)$, i.e. a PPTM who will attempt to compute $\llbracket v \rrbracket^{\eta, \rho}$, using the adversarial part ρ_a of the random tape ρ .

$\mathcal{A}$ is a PPTM, and since u is computable in polytime (and thus of polynomial size), and o_1, o_2 return hashes, of polynomial size as well, there exists a polynomial $q(\eta)$ bounding $\mathcal{A}(1^{\eta}, \rho_a, \llbracket u \rrbracket^{\eta, \rho}, \llbracket o_i \rrbracket^{\eta, \rho})$ for $i = 1, 2$.

Let $p_i(\eta) = \Pr_{\rho} [\mathcal{A}(1^{\eta}, \rho_a, \llbracket u \rrbracket^{\eta, \rho}, \llbracket o_i \rrbracket^{\eta, \rho}) = \llbracket v \rrbracket^{\eta, \rho}]$, for $i = 1, 2$. We know, by the first premise of the rule, that $p_2(\eta) \in \text{negl}(\eta)$, and must show that $p_1(\eta) \in \text{negl}(\eta)$.

We define the following two adversarial machines, taking the same inputs as $\mathcal{A}$:

- $\mathcal{A}_j$ simulates $\mathcal{A}$ until its j -th oracle call, and returns the input of that call (or a default value empty if there is no such call);

- $\mathcal{B}$ samples j uniformly between 1 and $q(\eta)$, independently of the randomness used by $\mathcal{A}$ (i.e. using a part of ρ_a further away). Then, $\mathcal{B}$ runs $\mathcal{A}_j$.

We then study the following events:

- $\phi^{\rho, \eta}(b)$: $\llbracket o_1 \rrbracket^{\eta, \rho}(b) \neq \llbracket o_2 \rrbracket^{\eta, \rho}(b)$, where b is a bitstring;
- $\Psi_i^{\rho, \eta}(\mathcal{M})$: $\phi^{\rho, \eta}(\mathcal{M}(1^{\eta}, \rho_a, \llbracket u \rrbracket^{\eta, \rho}, \llbracket o_i \rrbracket^{\eta, \rho}))$, where $\mathcal{M}$ is a PPTM and $i \in \{1, 2\}$;
- $E^{\rho, \eta}$: $\mathcal{A}(1^{\eta}, \rho_a, \llbracket u \rrbracket^{\eta, \rho}, \llbracket o_1 \rrbracket^{\eta, \rho})$ at some point calls its oracle on a bitstring b such that $\phi^{\rho, \eta}(b)$;
- $E_j^{\rho, \eta}$: $\mathcal{A}(1^{\eta}, \rho_a, \llbracket u \rrbracket^{\eta, \rho}, \llbracket o_1 \rrbracket^{\eta, \rho})$ makes its j -th oracle call on a bitstring b such that $\phi^{\rho, \eta}(b)$, and the j -th call is the first that satisfies this condition;
- $C_j^{\rho, \eta}$: $\mathcal{B}$ draws j at random in its first step.

Let us make several remarks, on some implications between these events, for fixed ρ and η :

- (1) $E^{\rho, \eta} \Rightarrow \exists! j \in [1, q(\eta)]. E_j^{\rho, \eta}$: indeed, if there is a faulty oracle call, it must be at most the $q(\eta)$ -th, and there is a unique first one. Note that the witness j may depend on ρ and η .
- (2) $C_j^{\rho, \eta}$ is independent of $\Psi_{\beta}^{\rho, \eta}(\mathcal{A}_k)$ for all k : by design, they depend on disjoint parts of the random tape ρ .
- (3) $\Psi_{\beta}^{\rho, \eta}(\mathcal{A}_j) \Rightarrow E^{\rho, \eta}$, for all j , which we prove by induction on j . Fix j such that $\Psi_{\beta}^{\rho, \eta}(\mathcal{A}_j)$. Then the input given by $\mathcal{A}(1^{\eta}, \rho_a, \llbracket u \rrbracket^{\eta, \rho}, \llbracket o_{\beta} \rrbracket^{\eta, \rho})$ in its j -th call to o_{β} satisfies $\phi^{\rho, \eta}$. Either it is the first such faulty call, or not.

If it is the first, then until that point the two oracles behave the same, and thus

$$\mathcal{A}_j(1^{\eta}, \rho_a, \llbracket u \rrbracket^{\eta, \rho}, \llbracket o_1 \rrbracket^{\eta, \rho}) = \mathcal{A}_j(1^{\eta}, \rho_a, \llbracket u \rrbracket^{\eta, \rho}, \llbracket o_2 \rrbracket^{\eta, \rho})$$

Hence, $\Psi_{\beta}^{\rho, \eta}(\mathcal{A}_j)$ implies that $E_j^{\rho, \eta}$ holds, and therefore so does $E^{\rho, \eta}$.

On the other hand, if it is not the first faulty call, then there exists a smaller j' such that $\Psi_{\beta}^{\rho, \eta}(\mathcal{A}_{j'})$, and we conclude by induction.

- (4) $E_j^{\rho, \eta} \Rightarrow \Psi_{\beta}^{\rho, \eta}(\mathcal{A}_j)$: this relies on the same inductive argument as the previous observation. If the first faulty call to o_1 by $\mathcal{A}(1^{\eta}, \rho_a, \llbracket u \rrbracket^{\eta, \rho}, \llbracket o_1 \rrbracket^{\eta, \rho})$ is the j -th, then until that point the two oracles behave the same, and thus that call is faulty as well for $\mathcal{A}(1^{\eta}, \rho_a, \llbracket u \rrbracket^{\eta, \rho}, \llbracket o_{\beta} \rrbracket^{\eta, \rho})$; otherwise we conclude by induction.

The rest of the proof proceeds in two steps.

Step 1: $\Pr_{\rho} [E^{\rho, \eta}] \geq |p_1(\eta) - p_2(\eta)|$

If $E^{\rho, \eta}$ is false, then in the execution of $\mathcal{A}(1^{\eta}, \rho_a, \llbracket u \rrbracket^{\eta, \rho}, \llbracket o_1 \rrbracket^{\eta, \rho})$, o_1 is only called on values on which o_2 gives the same answer. Thus

$$\mathcal{A}(1^{\eta}, \rho_a, \llbracket u \rrbracket^{\eta, \rho}, \llbracket o_1 \rrbracket^{\eta, \rho}) = \mathcal{A}(1^{\eta}, \rho_a, \llbracket u \rrbracket^{\eta, \rho}, \llbracket o_2 \rrbracket^{\eta, \rho}),$$

and therefore

$$\Pr_{\rho} [\mathcal{A}(1^{\eta}, \rho_a, \llbracket u \rrbracket^{\eta, \rho}, \llbracket o_1 \rrbracket^{\eta, \rho}) \neq \mathcal{A}(1^{\eta}, \rho_a, \llbracket u \rrbracket^{\eta, \rho}, \llbracket o_2 \rrbracket^{\eta, \rho})] \leq \Pr_{\rho} [E^{\rho, \eta}].$$

We conclude the proof of step 1 by observing that

$$|p_1(\eta) - p_2(\eta)| \leq \Pr_{\rho} \left[\begin{array}{c} \mathcal{A}(1^{\eta}, \rho_a, \llbracket u \rrbracket^{\eta, \rho}, \llbracket o_1 \rrbracket^{\eta, \rho}) \\ \neq \mathcal{A}(1^{\eta}, \rho_a, \llbracket u \rrbracket^{\eta, \rho}, \llbracket o_2 \rrbracket^{\eta, \rho}) \end{array} \right].$$

Step 2: $\Pr_\rho [\Psi_{\beta}^{\rho, \eta}(\mathcal{B})] \geq q(\eta)^{-1} \cdot \Pr_\rho [E^{\rho, \eta}]$ Indeed:

$$\begin{aligned}
& \Pr_\rho [\Psi_{\beta}^{\rho, \eta}(\mathcal{B})] \\
&= \sum_{j=0}^{q(\eta)} \Pr_\rho [C_j^{\rho, \eta} \wedge \Psi_{\beta}^{\rho, \eta}(\mathcal{B})] \quad (\text{decomposition}) \\
&= \sum_{j=0}^{q(\eta)} \Pr_\rho [C_j^{\rho, \eta} \wedge \Psi_{\beta}^{\rho, \eta}(\mathcal{A}_j)] \quad (\text{def. of } \mathcal{B} \text{ and } C_j) \\
&= \sum_{j=0}^{q(\eta)} \Pr_\rho [C_j^{\rho, \eta}] \cdot \Pr_\rho [\Psi_{\beta}^{\rho, \eta}(\mathcal{A}_j)] \quad (\text{Remark 2}) \\
&= \sum_{j=0}^{q(\eta)} q(\eta)^{-1} \cdot \Pr_\rho [\Psi_{\beta}^{\rho, \eta}(\mathcal{A}_j)] \\
&= \sum_{j=0}^{q(\eta)} q(\eta)^{-1} \cdot \Pr_\rho [\Psi_{\beta}^{\rho, \eta}(\mathcal{A}_j) \wedge E_k^{\rho, \eta}] \quad (\text{Remark 3}) \\
&= \sum_{j=0}^{q(\eta)} \sum_{k=0}^{q(\eta)} q(\eta)^{-1} \cdot \Pr_\rho [\Psi_{\beta}^{\rho, \eta}(\mathcal{A}_j) \wedge E_k^{\rho, \eta}] \quad (\text{Remark 1}) \\
&\geq \sum_{j=0}^{q(\eta)} q(\eta)^{-1} \cdot \Pr_\rho [\Psi_{\beta}^{\rho, \eta}(\mathcal{A}_j) \wedge E_j^{\rho, \eta}] \quad (\text{keep the cases } j = k) \\
&= \sum_{j=0}^{q(\eta)} q(\eta)^{-1} \cdot \Pr_\rho [E_j^{\rho, \eta}] \quad (\text{Remark 4}) \\
&= q(\eta)^{-1} \cdot \Pr_\rho [E^{\rho, \eta}] \quad (\text{Remark 1})
\end{aligned}$$

Conclusion: By these two steps:

$$|p_1(\eta) - p_2(\eta)| \leq q(\eta) \cdot \Pr_\rho [\Psi_{\beta}^{\rho, \eta}(\mathcal{B})].$$

Since $\mathcal{B}$ is an adversarial machine, by the second premise of the rule, the value it produces (given access to o_β) has a low probability of producing a different behavior between o_1 and o_2 : $\Pr_\rho [\Psi_{\beta}^{\rho, \eta}] \in \text{negl}(\eta)$, and therefore $|p_1(\eta) - p_2(\eta)| \in \text{negl}(\eta)$. Since p_2 is negligible, so is p_1 , which concludes the proof. $\square$

C.2 Probabilistic and Cryptographic Rules

We present our full set of probabilistic and cryptographic rules in Fig. 11, except PRF.R and PRF.L that are detailed in Appendix C.3.1. The rule are written in full, including the polytime simulatability conditions $\mathcal{E}; \emptyset \vdash_{\text{pptm}} \cdot$ (taken from [8]) for the cryptographic rules – the probabilistic rules rely on information-theoretic arguments, and require no such conditions.

We can easily design similar rules for other cryptographic assumptions. For example, the EUF-SIGN could be adapted to deal with a MAC instead of a signature.

DH rules. We present two local DH rules in Fig. 12, encoding the CDH and GDH assumptions. These rules are a novel, albeit minor, contribution of this paper. We give below the formal definitions of their syntactic side condition ψ , relying on the notion of generalized subterms $\mathcal{ST}_{\mathcal{E}}(t)$ as introduced in [8].

Given a term u , a name a and a generator g , we define $\mathcal{ST}_{\mathcal{E}, g, a}^{cdh}(u)$ recursively as $\mathcal{ST}_{\mathcal{E}}(u)$ up to one exception:

$$\mathcal{ST}_{\mathcal{E}, g, a}^{cdh}(g^a) = \emptyset$$

Then, for every model $\mathbb{M} : \mathcal{E}$ of Θ , every $\eta \in \mathbb{N}$ and every $\rho \in \mathbb{T}_{\mathbb{M}, \eta}$, we require that

$$\begin{aligned}
& \text{if } \llbracket \psi_{\text{CDH}}^{g, a}(u) \rrbracket_{\mathbb{M}; \mathcal{E}}^{\eta, \rho} = 1 \text{ then} \\
& \text{for every } (\vec{\alpha}, \psi, a) \in \mathcal{ST}_{\mathcal{E}, g, a}^{cdh}(u), \quad \llbracket \forall \vec{\alpha}. \neg \psi \rrbracket_{\mathbb{M}; \mathcal{E}}^{\eta, \rho} = 1.
\end{aligned}$$

For GDH, given a term u , names a, b and a generator g , we define $\mathcal{ST}_{\mathcal{E}, g, a, b}^{gdh}(u)$ recursively as $\mathcal{ST}_{\mathcal{E}}(u)$, with some exceptions:

$$\bullet \mathcal{ST}_{\mathcal{E}, g, a, b}^{gdh}(g^a) = \emptyset$$

$$\begin{aligned}
& \bullet \mathcal{ST}_{\mathcal{E}, g, a, b}^{gdh}(g^b) = \emptyset \\
& \bullet \mathcal{ST}_{\mathcal{E}, g, a, b}^{gdh}(\lambda x y. x = y^a) = \emptyset \\
& \bullet \mathcal{ST}_{\mathcal{E}, g, a, b}^{gdh}(\lambda x y. x = y^b) = \emptyset
\end{aligned}$$

Then, for every model $\mathbb{M} : \mathcal{E}$ of Θ , every $\eta \in \mathbb{N}$ and every $\rho \in \mathbb{T}_{\mathbb{M}, \eta}$, we require that

$$\begin{aligned}
& \text{if } \llbracket \psi_{\text{GDH}}^{g, a, b}(u) \rrbracket_{\mathbb{M}; \mathcal{E}}^{\eta, \rho} = 1 \\
& \text{then for every } (\vec{\alpha}, \psi, a) \in \mathcal{ST}_{\mathcal{E}, g, a, b}^{gdh}(u) \\
& \quad \text{or } (\vec{\alpha}, \psi, b) \in \mathcal{ST}_{\mathcal{E}, g, a, b}^{gdh}(u), \\
& \text{we have } \llbracket \forall \vec{\alpha}. \neg \psi \rrbracket_{\mathbb{M}; \mathcal{E}}^{\eta, \rho} = 1.
\end{aligned}$$

By ensuring that names a and b only occur as g^a, g^b (or as a DDH oracle for GDH), we show the soundness of the two rules by reduction to the respective cryptographic assumption CDH or GDH. These reductions follow the strategy laid out in [8] (we omit the details here).

Soundness. We show the soundness of our probabilistic and cryptographic rules.

THEOREM C.2. *The rules of Fig. 11 are sound.*

PROOF. **FRESHR:**

We use the local rule for Fresh from [8], that we recall below:

$$\frac{\text{L.FRESH} \quad n \text{ has type } \tau \quad \text{large} \in \text{label}(\tau)}{\mathcal{E}; \Theta; \Gamma \vdash u = n \Rightarrow \neg \psi_{\text{fresh}}^n(u)}$$

To prove **FRESHR**, assume by contradiction that we have a function f such that:

$$\mathcal{E}; \Theta \vdash \text{adv}(f) \quad \mathcal{E}; \Theta \vdash \neg [f(u) \neq n].$$

Introducing the negation, we get:

$$\mathcal{E}; \Theta, [f(u) \neq n] \vdash \perp \quad (3)$$

The premises over the type of n from **FRESHR** ensure that we can apply **L.FRESH**. Thus, taking the contrapositive of **L.FRESH** and moving to a global sequent:

$$\mathcal{E}; \Theta \vdash [\psi_{\text{fresh}}^n(f(u)) \Rightarrow f(u) \neq n].$$

Applying this on the hypothesis in Eq. (3), we get:

$$\mathcal{E}; \Theta, [\psi_{\text{fresh}}^n(f(u))] \vdash \perp$$

To conclude, it is sufficient to show that

$$\psi_{\text{fresh}}^n(f(u)) \text{ and } \psi_{\text{fresh}}^n(u) \quad (4)$$

are the same formula.

This requires discussing some details that we omitted while recalling the **L.FRESH** rule and the ψ -formulas from [8]. In this rule, the formula $\psi_{\text{fresh}}^n(w)$ (for some term w) is *any* formula ensuring that the name n is fresh in w – we do explain how this freshness semantical condition is formally defined, see [8] for details. This means that we have some leeway in choosing the formula $\psi_{\text{fresh}}^n(w)$ when applying the rule, as long as this formula ensures the semantic freshness of n – further, [8] provides an explicit procedure allowing to build such a formula, making the rule effective. The semantic condition ensuring that n is fresh in w is based on a

$$\begin{array}{c}
\text{FRESHR} \\
\frac{n \text{ has type } \tau \quad \text{large} \in \text{label}(\tau) \quad \mathcal{E}; \Theta \vdash [\psi_{\text{fresh}}^n(u)]}{\mathcal{E}; \Theta \vdash u \not\vdash n}
\end{array}
\quad
\begin{array}{c}
\text{FRESHL} \\
\frac{\mathcal{E}; \Theta \vdash u \not\vdash v \quad \mathcal{E}; \Theta \vdash [\psi_{\text{fresh}}^n(u, v)]}{\mathcal{E}; \Theta \vdash u, n \not\vdash v}
\end{array}$$

$$\begin{array}{c}
\text{CDH} \\
\frac{\mathcal{E}; \Theta \vdash [\psi_{\text{CDH}}^{g,a}(u)] \quad \mathcal{E}; \Theta \vdash [\psi_{\text{CDH}}^{g,b}(u)] \quad \mathcal{E}; \Theta; \emptyset \vdash_{\text{pptm}} u}{\mathcal{E}; \Theta \vdash u \not\vdash g^{a,b}}
\end{array}
\quad
\begin{array}{c}
\text{GDH} \\
\frac{\mathcal{E}; \Theta \vdash [\psi_{\text{GDH}}^{g,a,b}(u)] \quad \mathcal{E}; \Theta; \emptyset \vdash_{\text{pptm}} u}{\mathcal{E}; \Theta \vdash u \not\vdash g^{a,b}}
\end{array}
\quad
\begin{array}{c}
\text{PRF-SIMPLE} \\
\frac{\mathcal{E}; \Theta \vdash [\psi_{\text{PRF,key}}^k(u, v, m)] \quad \mathcal{E}; \Theta \vdash [\psi_{\text{PRF,msg}}^{m,k}(u, v, m)] \quad \mathcal{E}; \Theta; \emptyset \vdash_{\text{pptm}} u, m, v}{\mathcal{E}; \Theta \vdash u, h(m, k) \not\vdash v}
\end{array}$$

$$\begin{array}{c}
\text{EUF-SIGN} \\
\frac{\mathcal{E}; \Theta \vdash [\psi_{\text{EUF,skey}}^k(u, m)] \quad \mathcal{E}; \Theta \vdash [\psi_{\text{EUF,sign}}^{k,m}(u, m)] \quad \mathcal{E}; \Theta; \emptyset \vdash_{\text{pptm}} u, m}{\mathcal{E}; \Theta \vdash u \not\vdash \text{Sign}(m, k)}
\end{array}$$

In the rules above, n, a, b , and k are names. For the sake of simplicity, we present the rules for un-indexed names. We asked that any declared variable v in $\mathcal{E}$ occurs only in η -long form, and is either a name or such that $\mathcal{E}; \Theta \vdash \text{adv}(v)$.

Figure 11: Probabilistic and cryptographic rules for $\not\vdash$ (unabridged).

$$\begin{array}{c}
\text{L.CDH} \\
\frac{\mathcal{E}; \Theta; \emptyset \vdash_{\text{pptm}} u}{\mathcal{E}; \Theta; \Gamma \vdash u = g^{a,b} \Rightarrow \neg(\psi_{\text{CDH}}^{g,a}(u) \wedge \psi_{\text{CDH}}^{g,b}(u))}
\end{array}$$

$$\begin{array}{c}
\text{L.GDH} \\
\frac{\mathcal{E}; \Theta; \emptyset \vdash_{\text{pptm}} u}{\mathcal{E}; \Theta; \Gamma \vdash u = g^{a,b} \Rightarrow \neg \psi_{\text{GDH}}^{g,a,b}(u)}
\end{array}$$

In the rules above, a and b are names. For the sack of simplicity, we present the rules for un-indexed names. We ask that any declared variable v in $\mathcal{E}$ occurs only in η -long form, and is either a name or such that $\mathcal{E}; \Theta \vdash \text{adv}(v)$.

Figure 12: Local rules for CDH and GDH.

generalization of the sub-terms of w which ignores the application of a top-level $\text{adv}(\cdot)$ symbol to w . Thus, any acceptable formula $\psi_{\text{fresh}}^n(\cdot)$ for w is also acceptable for $f(w)$ when $\text{adv}(f)$.

Consequently, we use the formula $\psi_{\text{fresh}}^n(u)$ when applying the **L.FRESH** rule to $f(u)$. Thus, the two formulas in Eq. (4) are identical, which concludes this proof.

FRESHL:

We use the equivalence rule for Fresh of [8], that we recall below:

$$\begin{array}{c}
\text{G.EQUIV.FRESH} \\
\frac{\mathcal{E}; \Theta \vdash [\psi_{\text{fresh}}^n(u)]}{\mathcal{E}; \Theta \vdash u, n \sim u, n_{\text{fresh}}}
\end{array}$$

First, consider a name symbol n_{fresh} that does does appear in u nor in the definitions in $\mathcal{E}$. By hypothesis of **FRESHL**, we know that $u \not\vdash v$. First, we show that this implies that $u, n_{\text{fresh}} \not\vdash v$ by contradiction. Assume f computing v from u, n_{fresh} with non-negligible probability, we are going to build f' violating the hypothesis $u \not\vdash v$. To do that, we let f' be the function which on input u simulates the sampling of n_{fresh} from the adversarial randomness and then runs f . We can check that f' behaves as f , in the sense that it computes v with the same probability.²

Thus, we know that $u, n_{\text{fresh}} \not\vdash v$. Using **G.EQUIV.FRESH** over the sequence u, v, n , we have:

$$u, v, n \sim u, v, n_{\text{fresh}}.$$

We conclude by using **Rw-EQUIV** on the indistinguishability above and on the secrecy property $u, n_{\text{fresh}} \not\vdash v$, which yields $u, n \not\vdash v$.

PRF-SIMPLE:

We use the equivalence rule for PRF of [8], that we recall below:

$$\begin{array}{c}
\text{G.PRIF} \\
\frac{\mathcal{E}; \Theta \vdash [\psi_{\text{PRF,key}}^k(u, m)] \quad \mathcal{E}; \Theta \vdash [\psi_{\text{PRF,msg}}^{m,k}(u, m)] \quad \mathcal{E}; \Theta \vdash [\psi_{\text{fresh}}^n(u, m)] \quad \mathcal{E}; \Theta; \emptyset \vdash_{\text{pptm}} u, m}{\mathcal{E}; \Theta \vdash u, h(m, k) \sim u, n}
\end{array}$$

We apply this rule with $u' = (u, v)$ and n a name that does not occur in u, v and m , nor in any definition in $\mathcal{E}$. The premises of **PRF-SIMPLE** imply those of **G.PRIF**, so that $u, h(m, k), v \sim u, n, v$. With the rule **Rw-EQUIV**, we get $u, h(m, k) \not\vdash v$ if $u, n \not\vdash v$. Finally, with **FRESHL**, we prove that the premise $u \not\vdash v$ implies $u, n \not\vdash v$.

EUF-SIGN, CDH, GDH: Those rules have proofs which are similar to the proof of **FRESHR**, i.e. we apply a local rule to a term obtained by negating the non-deduction predicate under consideration. Thus, **EUF-SIGN** follows from the rule **L.EUF** in [8], and **CDH** and **GDH** follow from the **L.CDH** and **L.GDH** in Fig. 12. $\square$

C.3 PRF Rules

We present here the full formalization of the PRF rules, as well as their proofs. These complete rules use an indexed key kt , instead of a simple key k , adding several hypothesis on this index t and its type. We show the left PRF rule **PRF.L** in Appendix C.3.1. The variants **PRF.L-EQ** and **PRF.L-SEC** are argued for in Remark 1 and the right PRF rule **PRF.R** in Appendix C.3.2.

²There is a subtlety here: we must ensure that f' has enough fresh randomness at its disposal in the adversarial tape. This can be done by defining f' in a new model in which we extended the size of the adversarial tape as needed. Then, since the secrecy property $u \not\vdash v$ is assumed to hold in any model, we can apply consider it in the extended model.

C.3.1 Left rule.

PRF.L

$$\frac{\begin{array}{c} t \text{ has type } \tau \quad \tau \text{ is serializable} \\ \mathcal{E}; \Theta \vdash \text{adv}(t) \quad \mathcal{E}; \Theta; \emptyset \vdash_{\text{pptm}} u, v, m \quad \mathcal{E}; \Theta \vdash u, o_h^k \not\vdash v \\ \mathcal{E}; \Theta \vdash [\psi_{\text{PRF,key}}^{k,t}(u, v, m)] \quad \mathcal{E}; \Theta \vdash [\psi_{\text{PRF,msg}}^{m,k}(u, v)] \end{array}}{\mathcal{E}; \Theta \vdash u, o_h^k, h(m, k(t)) \not\vdash v}$$

Note that:

- $\psi_{\text{PRF,msg}}^{m,k}(m)$ is not required, i.e. m can be hashed to compute m itself;
- τ needs to be encodable as bitstrings of length polynomial in η , and t must be adversarial, so that an adversary who wishes to evaluate a term encounters “ $k(t')$ ” can easily check whether $t = t'$;
- no syntactic conditions are checked on the oracle o_h^k , it is handled in an ad-hoc way by the rule.

PROOF. By contradiction, assume the rule is unsound. Consider a model $\mathbb{M}$ for which the premises of the rules are satisfied, and such that $\mathbb{M} \models \Theta$, but where the conclusion of the rule does not hold. That is, by definition of $\not\vdash$, the formula $\check{v}f. \text{adv}(f) \Rightarrow [f(u, o_h^k, h(m, k(t))) \neq v]$ does not hold in $\mathbb{M}$. Thus, there exists an adversary $\mathcal{A}_3 \in \text{PPTMs}$ that computes v from u , the oracle o_h , and the hash:

$$\Pr_{\rho} \left[\mathcal{A}_3^{o_h}(1^\eta, \rho_a, [u]_{\mathbb{M}}^{\eta,\rho}, [h(m, k(t))]_{\mathbb{M}}^{\eta,\rho}) = [v]_{\mathbb{M}}^{\eta,\rho} \right] \notin \text{negl}(\eta). \quad (5)$$

Note that

- (1) We call this adversary $\mathcal{A}_3$ for reasons that will become clear later on in the proof – it will be the third machine called at some point.
- (2) We write $\mathcal{A}_3^{o_h}$ to specify that $\mathcal{A}_3$ can call the function $[o_h^k]_{\mathbb{M}}^{\eta,\rho}$ as an oracle.
- (3) We may write later on $\mathcal{A}_3(\cdot, \rho, \cdot, \cdot)$ rather than $\mathcal{A}_3(\cdot, \rho_a, \cdot, \cdot)$ for simplicity, but $\mathcal{A}_3$ still only accesses the adversarial part ρ_a of the tape ρ .

Consider the PRF game:

$\text{Exp}_{\mathcal{B}}^{\text{prf},\beta}(1^\eta, \rho)$	$O_0(w)$	$O_1(w)$
$k_0 \leftarrow \rho$	return $[h]_{\mathbb{M}}^{\eta,\rho}(1^\eta, w, k_0)$	if $L[w] = \perp$ then
$L \leftarrow \perp$		$L[w] \leftarrow \{0, 1\}^\eta$
$\beta' \leftarrow \mathcal{B}^{O_\beta}(1^\eta, \rho)$		return $L[w]$
return β'		

We adopt an early sampling point of view, and write $k_0 \leftarrow \rho$ to represent reading k_0 from the relevant part (of length η) of the random tape ρ . $\mathcal{B}$ can access all of ρ , except for the part used to read k_0 .

The advantage of $\mathcal{B}$ is

$$\text{Adv}_{\mathcal{B}}^{\text{prf}}(\eta) = \left| \Pr_{\rho} \left[\text{Exp}_{\mathcal{B}}^{\text{prf},0}(1^\eta, \rho) = 1 \right] - \Pr_{\rho} \left[\text{Exp}_{\mathcal{B}}^{\text{prf},1}(1^\eta, \rho) = 1 \right] \right|.$$

The PRF assumption states that any PPTM $\mathcal{B}$ has a negligible advantage. Note that it does not depend on where precisely k_0 is read on ρ , as long as $\mathcal{B}$ is not given access to that part of ρ .

Since t is an adversarial bitstring, its value depends on η and the adversarial part of ρ . In particular, it does not depend on the value $[k(t')]_{\mathbb{M}}^{\eta,\rho}$ for any t' , which are in the honest part of ρ . So, the value $[k(t)]_{\mathbb{M}}^{\eta,\rho}$ is independent of $[t]_{\mathbb{M}}^{\eta,\rho}$. We can then consider the instance of the PRF game where $k_0 = [k(t)]_{\mathbb{M}}^{\eta,\rho}$ as long as the attacker cannot access $[k(t)]_{\mathbb{M}}^{\eta,\rho}$ for any ρ, η .

Let us use $\mathcal{A}_3$ to construct an adversary $\mathcal{B}$ that wins the PRF game.

High-level intuition. Roughly, $\mathcal{B}$ proceeds as follows. First, it attempts to compute u, v , and $h(m, k(t))$, using oracle O_β as needed to compute hashes with $k(t)$. Then, it calls $\mathcal{A}_3$ on u, o_h and $h(m, k(t))$. o_h is simulated by $\mathcal{B}$ by forwarding all queries from $\mathcal{A}_3$ to the oracle provided by the PRF game. Finally, $\mathcal{B}$ checks whether $\mathcal{A}_3$ returned v . If so, it answers $\beta' = 1$, and otherwise $\beta' = 0$.

If $\mathcal{B}$ uses the real hashing oracle O_0 , it correctly simulates

$$\mathcal{A}_3^{o_h}(1^\eta, \rho_a, [u]_{\mathbb{M}}^{\eta,\rho}, [h(m, k(t))]_{\mathbb{M}}^{\eta,\rho}) \quad \text{and} \quad [v]_{\mathbb{M}}^{\eta,\rho},$$

and thus answers 1 with the same probability as $\mathcal{A}_3$'s in (5), which is non-negligible. However, if $\mathcal{B}$ uses the idealised oracle O_1 , the hash $\mathcal{B}$ gives to $\mathcal{A}_3$ is essentially a random value, chosen (depending on m) among those produced by the oracle O_1 . We will show that, if $\mathcal{A}_3$ could efficiently compute v from u, O_1 , and the idealised hash, then it could as well simulate the hash by drawing its value at random. This would allow computing v from just u and O_1 , which we will show contradicts the premise that $u, o_h \not\vdash v$. This will establish that, when $\beta = 1$, $\mathcal{B}$ only answers 1 with negligible probability, and thus that $\mathcal{B}$ wins the PRF game.

Defining $\mathcal{B}$. Let us now define $\mathcal{B}$ formally. Since $\mathcal{E}; \Theta; \emptyset \vdash_{\text{pptm}} u, v, m$ and $\mathcal{E}; \Theta; \emptyset \vdash \psi_{\text{PRF,key}}^{k,t}(u, v, m)$, then we know that there exists a PPTM $\mathcal{A}_1^{O_0}(1^\eta, \rho)$ that computes $[u]_{\mathbb{M}}^{\eta,\rho}, [v]_{\mathbb{M}}^{\eta,\rho}$, and in doing so never needs to access the part of ρ associated with $k(t)$, and calls the hashing oracle O_0 when a hash with that key is needed. Similarly, there exists a PPTM $\mathcal{A}_2^{O_0}(1^\eta, \rho)$, that computes $[m]_{\mathbb{M}}^{\eta,\rho}$.

We have:

$$\Pr_{\rho} \left[\begin{array}{l} \mathcal{A}_1^{O_0}(1^\eta, \rho) = [u]_{\mathbb{M}}^{\eta,\rho}, [v]_{\mathbb{M}}^{\eta,\rho} \\ \wedge \mathcal{A}_2^{O_0}(1^\eta, \rho) = [m]_{\mathbb{M}}^{\eta,\rho} \end{array} \right] \geq \Pr_{\rho} \left[[\psi_{\text{PRF,key}}^{k,t}(u, v, m)]_{\mathbb{M}}^{\eta,\rho} \right]$$

As well as:

$$\Pr_{\rho} \left[[\psi_{\text{PRF,key}}^{k,t}(u, v, m)]_{\mathbb{M}}^{\eta,\rho} \right] \in \text{ow}(\eta)$$

More precisely, these machines recursively evaluate the terms. The only thing they are not allowed to compute is $k(t)$: since the predicate $\psi_{\text{PRF,key}}^{k,t}(u, v, m)$ holds overwhelmingly, this value only occurs as a hashing key. When encountering a hash with $k(t')$, they thus evaluate $[t']_{\mathbb{M}}^{\eta,\rho}$. Since t is adversarial, they can also evaluate $[t]_{\mathbb{M}}^{\eta,\rho}$. Both of these values can be seen as bitstrings of polynomial length, by the condition that τ is serializable, and can thus easily be compared. The machines then call the hashing oracle if they are equal, or, otherwise, compute the key (which is then not $k(t)$) and the hash themselves.

The adversary $\mathcal{B}$ is then defined as follows:

$$\begin{aligned} & \frac{\mathcal{B}^{O_\beta}(1^\eta, \rho)}{\tilde{u}, \tilde{v} \leftarrow \mathcal{A}_1^{O_\beta}(1^\eta, \rho)} \\ & \tilde{m} \leftarrow \mathcal{A}_2^{O_\beta}(1^\eta, \rho) \\ & \tilde{h} \leftarrow O_\beta(\tilde{m}) \\ & \text{return } (\tilde{h} = \mathcal{A}_3^{O_\beta}(1^\eta, \rho, \tilde{u}, \tilde{h})) \end{aligned}$$

$\mathcal{B}$ executes machines $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$, using its oracle O_β to answer hash oracle calls, and the tape ρ to provide all randomness except the hashing key – which the three machines never ask for by construction.

Without loss of generality, we will from now on assume that, on any input, each of the machines $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$ never calls its oracle on the same value twice – this is always feasible by memorising the previous oracle response to replace future calls. Moreover, we will assume that $\mathcal{A}_2$, on any input, always makes during its execution a call to its oracle on the value $\tilde{m}$ it returns – it can always make an additional call just before returning it if that is not the case.

Proving that $\mathcal{B}$ wins against PRF. If $\beta = 0$, the simulators $\mathcal{A}_1^{O_0}, \mathcal{A}_2^{O_0}$ are run as expected, and the oracle O_0 given to $\mathcal{A}_3$ behaves exactly as o_h . Thus

$$\left| \Pr_\rho \left[\text{Exp}_{\mathcal{B}}^{\text{prf},0}(1^\eta, \rho) = 1 \right] - \Pr_\rho \left[\mathcal{A}_3^{O_h}(1^\eta, \rho, \llbracket u \rrbracket_{\mathbb{M}}^{\eta, \rho}, \llbracket h(m, k(t)) \rrbracket_{\mathbb{M}}^{\eta, \rho}) = \llbracket v \rrbracket_{\mathbb{M}}^{\eta, \rho} \right] \right| \in \text{negl}(\eta)$$

which means, since $\mathcal{A}_3$ has a non-negligible advantage, that

$$\Pr_\rho \left[\text{Exp}_{\mathcal{B}}^{\text{prf},0}(1^\eta, \rho) = 1 \right] \notin \text{negl}(\eta).$$

Consider now the case $\beta = 1$. We will now show, by a series of game hops, that

$$\Pr_\rho \left[\text{Exp}_{\mathcal{B}}^{\text{prf},1}(1^\eta, \rho) = 1 \right] \in \text{negl}(\eta),$$

which naturally implies, together with the previous case, that $\mathcal{B}$ wins the PRF game with a non-negligible advantage.

Initial game: We consider the following game $\mathcal{G}_0$:

$$\begin{aligned} & \frac{\mathcal{G}_0(1^\eta)}{\rho \leftarrow \mathfrak{s}} \\ & \tilde{u}, \tilde{v} \leftarrow \mathcal{A}_1^{O_1}(1^\eta, \rho) \\ & \tilde{m} \leftarrow \mathcal{A}_2^{O_1}(1^\eta, \rho) \\ & \tilde{h} \leftarrow O_1(\tilde{m}) \\ & \text{return } \tilde{v} = \mathcal{A}_3^{O_1}(1^\eta, \rho, \tilde{u}, \tilde{h}) \end{aligned}$$

$\mathcal{G}_0$ is in fact the right side of the PRF game with $\mathcal{B}$:

$$\Pr_\rho \left[\text{Exp}_{\mathcal{B}}^{\text{prf},1}(1^\eta, \rho) = 1 \right] = \Pr \left[\mathcal{G}_0(1^\eta, \rho) \right].$$

First hop: We first exclude the possibility of $\mathcal{A}_1$ calling the oracle O_1 on $\tilde{m}$.

We consider the game $\mathcal{G}_1$:

$$\begin{aligned} & \frac{\mathcal{G}_1(1^\eta)}{\rho \leftarrow \mathfrak{s}} \\ & \tilde{u}, \tilde{v} \leftarrow \mathcal{A}_1^{O_1}(1^\eta, \rho) \\ & \tilde{m} \leftarrow \mathcal{A}_2^{O_1}(1^\eta, \rho) \\ & \tilde{h} \leftarrow O_1(\tilde{m}) \\ & \text{return } (\tilde{v} = \mathcal{A}_3^{O_1}(1^\eta, \rho, \tilde{u}, \tilde{h})) \wedge \neg \text{bad} \end{aligned}$$

where bad is defined as follows. Let log denote the log of all values on which O_1 is queried by $\mathcal{A}_1$ during the experiment, either during its execution, or to evaluate first-order functions returned as closures as part of $\tilde{u}$, that $\mathcal{A}_3$ may later call during its own execution. bad is then defined as the condition $\tilde{m} \in \text{log}$. bad is tied to a given experiment, here $\mathcal{G}_1$ – e.g., if $\mathcal{A}_2$ or $\mathcal{A}_3$ were called on different parameters, then the value of $\tilde{m}$ and the oracle's log could differ, and thus so could bad. When there is a potential ambiguity, we will write $\text{bad}_{\mathcal{G}}$ to specify that $\tilde{m}$ and log are to be understood w.r.t. the execution of a game $\mathcal{G}$. Note for instance that $\text{bad}_{\mathcal{G}_0} = \text{bad}_{\mathcal{G}_1}$.

We have

$$\Pr \left[\mathcal{G}_0(1^\eta) \right] \leq \Pr \left[\mathcal{G}_1(1^\eta) \right] + \Pr \left[\text{bad}_{\mathcal{G}_0} \right].$$

Besides, consider the game $\mathcal{G}_0^\beta$ for $\beta \in \{0, 1\}$:

$$\begin{aligned} & \frac{\mathcal{G}_0^\beta(1^\eta)}{\rho \leftarrow \mathfrak{s}} \\ & \tilde{u}, \tilde{v} \leftarrow \mathcal{A}_1^{O_\beta}(1^\eta, \rho) \\ & \tilde{m} \leftarrow \mathcal{A}_2^{O_\beta}(1^\eta, \rho) \\ & \tilde{h} \leftarrow O_\beta(\tilde{m}) \\ & \_ \leftarrow \mathcal{A}_3^{O_\beta}(1^\eta, \rho, \tilde{u}, \tilde{h}) \\ & \text{return bad} \end{aligned}$$

$(\mathcal{G}_0^0, \mathcal{G}_0^1)$ is itself an instance of the PRF game, and therefore the associated advantage $\text{Adv}_0^{\text{prf}} = \left| \Pr \left[\text{bad}_{\mathcal{G}_0^1} \right] - \Pr \left[\text{bad}_{\mathcal{G}_0^0} \right] \right|$ is negligible.

In addition, we know, by construction of $\psi_{\text{PRF}, \text{msg}}^{m, k}(u, v)$, that

$$\Pr \left[\text{bad}_{\mathcal{G}_0^0} \right] \leq \Pr_\rho \left[\neg \llbracket \psi_{\text{PRF}, \text{msg}}^{m, k}(u, v) \rrbracket_{\mathbb{M}}^{\eta, \rho} \right].$$

Since $\text{bad}_{\mathcal{G}_0^1}$ is in fact the same as $\text{bad}_{\mathcal{G}_0}$, we get that:

$$\Pr \left[\text{bad}_{\mathcal{G}_0} \right] \leq \Pr_\rho \left[\neg \llbracket \psi_{\text{PRF}, \text{msg}}^{m, k}(u, v) \rrbracket_{\mathbb{M}}^{\eta, \rho} \right] + \text{Adv}_0^{\text{prf}}.$$

Therefore:

$$\Pr \left[\mathcal{G}_0(1^\eta) \right] \leq \Pr \left[\mathcal{G}_1(1^\eta) \right] + \Pr_\rho \left[\neg \llbracket \psi_{\text{PRF}, \text{msg}}^{m, k}(u, v) \rrbracket_{\mathbb{M}}^{\eta, \rho} \right] + \text{Adv}_0^{\text{prf}}.$$

Second hop: We now remove the possibility for $\mathcal{A}_3$ to call the oracle on $\tilde{m}$, which can be done by using directly $\tilde{h}$ instead. The subtlety is that $\mathcal{A}_3$ does not know the value of $\tilde{m}$, and thus does not know which input it should refrain from sending to the oracle: we will have to guess it.

For any message x , we let O'_x be the oracle O_1 punctured on input x :

$$O'_x(y) = \begin{cases} O_1(y) & \text{if } y \neq x \\ \perp & \text{otherwise} \end{cases}$$

We consider the game $\mathcal{G}_2$:

$$\begin{array}{l}
\mathcal{G}_2(1^\eta) \\
\rho \leftarrow \$ \\
\hline
\tilde{u}, \tilde{v} \leftarrow \mathcal{A}_1^{O_1}(1^\eta, \rho) \\
\tilde{m} \leftarrow \mathcal{A}_2^{O_1}(1^\eta, \rho) \\
\tilde{h} \leftarrow O_1(\tilde{m}) \\
\text{return } (\tilde{v} = \mathcal{A}_4^{O'_m}(1^\eta, \rho, \tilde{u}, \tilde{h})) \wedge \neg \text{bad}
\end{array}$$

where $\mathcal{A}_4$ is defined as follows. Since $\tilde{u}, \tilde{m}, \tilde{h}$ are produced by $\mathcal{A}_1$ followed by $\mathcal{A}_2$, which are PPTMs, and O_1 only returns values of length η , there exists a polynomial $p_1(\eta)$ that bounds their total length in any execution of $\mathcal{G}_1(1^\eta)$. Since $\mathcal{A}_3$ is a PPTM, there exists a polynomial $p_2(n)$ that strictly bounds the number of calls it makes to its oracle in any execution on an input of size n . Thus, the polynomial $p \triangleq p_2 \circ p_1$ bounds the number of calls made by $\mathcal{A}_3$ in any execution of $\mathcal{G}_1(1^\eta)$.

Let then $\mathcal{A}_4$ be the following machine:

$$\begin{array}{l}
\mathcal{A}_4^O(1^\eta, \rho, x, y) \\
i \leftarrow \$ [1, p(\eta)] \\
\hline
\text{run } \mathcal{A}_3^O(1^\eta, \rho, x, y) \text{ normally, except for its} \\
i\text{-th call to } O: \text{ do not make that call,} \\
\text{and use } y \text{ instead}
\end{array}$$

The definition of $\text{bad}_{\mathcal{G}_2}$ remains the same – now accounting for the calls made to functions returned in $\tilde{u}$ by $\mathcal{A}_4$, rather than $\mathcal{A}_3$.

Consider the execution of $\mathcal{A}_3$ in $\mathcal{G}_1$. Let then i_0 be the random variable defined as

- if $\mathcal{A}_3$ does attempt to call the oracle on $\tilde{m}$, then i_0 is the index of that call (i.e., it is the i_0 -th call made by $\mathcal{A}_3$);
- otherwise, $i_0 = p(\eta)$.

It is clear that, for any given ρ , if i is guessed such that $i = i_0$, then $\mathcal{A}_4$ in $\mathcal{G}_2$ accurately simulates $\mathcal{A}_3$ in $\mathcal{G}_1$: if $i_0 < p(\eta)$, answering the i_0 -th call with $\tilde{h}(= O_1(\tilde{m}))$ is correct, and all other calls are on values other than $\tilde{m}$ (recall that $\mathcal{A}_3$ makes by assumption no duplicate calls) and O'_m answers them accurately; otherwise, if $i_0 = p(\eta)$, then none of the oracle calls are intercepted (since p is a strict bound), and O'_m answers all calls like O_1 . Therefore:

$$\Pr[\mathcal{G}_2(1^\eta)] \geq \Pr[\mathcal{G}_2(1^\eta) \wedge i = i_0] = \Pr[\mathcal{G}_1(1^\eta) \wedge i = i_0].$$

The event “ $i = i_0$ ” is independent of the execution of $\mathcal{G}_1$, since i is a fresh uniformly sampled value that does not appear in $\mathcal{G}_1$, hence

$$\Pr[\mathcal{G}_1(1^\eta) \wedge i = i_0] = \Pr[i = i_0] \cdot \Pr[\mathcal{G}_1(1^\eta)].$$

For the same reason, i_0 and i are independent – i_0 is defined from $\mathcal{G}_1$, where i does not even exist, and i is uniformly sampled separately. Therefore, $\Pr[i = i_0] = p(\eta)^{-1}$. Altogether, we get:

$$\Pr[\mathcal{G}_1(1^\eta)] \leq p(\eta) \cdot \Pr[\mathcal{G}_2(1^\eta)].$$

Third hop: Our next objective will be to decompose $\mathcal{A}_2$'s execution, to isolate its call to O_1 on $\tilde{m}$ (which exists and is unique by assumption). There again, the subtlety is that $\mathcal{A}_2$ does not know *a priori* the value of $\tilde{m}$ before the end of its execution, and will have

to guess which of its call is the right one. This hop prepares the terrain, by isolating one oracle call picked at random.

As before, $\mathcal{A}_2$ is a PPTM, and thus there exists a polynomial $q(\eta)$ that bounds the number of calls $\mathcal{A}_2^{O_1}(1^\eta, \rho)$ makes to O_1 , for all ρ . We introduce the following game $\mathcal{G}_3$:

$$\begin{array}{l}
\mathcal{G}_3(1^\eta) \\
\rho \leftarrow \$ \\
\hline
j \leftarrow \$ [1, q(\eta)] \\
\tilde{u}, \tilde{v} \leftarrow \mathcal{A}_1^{O_1}(1^\eta, \rho) \\
\text{done, st, } \tilde{m}' \leftarrow \mathcal{A}_{2,j-1}^{O_1}(1^\eta, \rho) \\
\tilde{h}' \leftarrow \text{if } \neg \text{done then } O_1(\tilde{m}') \text{ else } \perp \\
\tilde{m} \leftarrow \mathcal{A}_{2,j+1}^{O_1}(1^\eta, \rho, \text{done, st, } \tilde{m}', \tilde{h}') \\
\tilde{h} \leftarrow O_1(\tilde{m}) \\
\text{return } (\tilde{v} = \mathcal{A}_4^{O'_m}(1^\eta, \rho, \tilde{u}, \tilde{h})) \wedge \neg \text{bad}
\end{array}$$

where $\mathcal{A}_{2,j-1}, \mathcal{A}_{2,j+1}$ represent the execution of $\mathcal{A}_2$ before its j -th call to O_1 , and after:

- $\mathcal{A}_{2,j-1}^{O_1}(1^\eta, \rho)$ is the machine that runs $\mathcal{A}_2^{O_1}(1^\eta, \rho)$, forwarding its oracle calls to O_1 , until it makes its j -th call, on some value $\tilde{m}'$. $\mathcal{A}_{2,j-1}$ does not forward that call, and instead stops, and returns the value $\tilde{m}'$, as well as a bit done set to 0, to indicate that $\mathcal{A}_2$'s computation is not over yet, and a state st containing the entire internal state of $\mathcal{A}_2$, so that it may be resumed later. If $\mathcal{A}_2$ does in fact not make j calls, i.e. if it terminates before that, $\mathcal{A}_{2,j-1}$ returns a bit done set to 1, and the value produced by $\mathcal{A}_2$ as $\tilde{m}'$.
- $\mathcal{A}_{2,j+1}^{O_1}(1^\eta, \rho, \text{done, st, } \tilde{m}', \tilde{h}')$ is the machine that, provided done = 0, resumes the execution of $\mathcal{A}_2$ from state st: in that state, $\mathcal{A}_2$ should be expecting a response from the oracle (if not, $\mathcal{A}_{2,j+1}$ just stops), $\mathcal{A}_{2,j+1}$ feeds it $\tilde{h}'$ as that response, and continues its execution, until it returns a value, which $\mathcal{A}_{2,j+1}$ returns. If done = 1, $\mathcal{A}_{2,j+1}$ instead immediately returns $\tilde{m}'$.

It is clear that, no matter what value is drawn for j , the execution of $\mathcal{A}_2$ simulated by $\mathcal{A}_{2,j-1}$ and $\mathcal{A}_{2,j+1}$ is accurate – we have only set aside the j -th oracle call, if it exists, but nothing has changed compared to $\mathcal{G}_2$. Therefore:

$$\Pr[\mathcal{G}_2(1^\eta)] = \Pr[\mathcal{G}_3(1^\eta)].$$

Fourth hop: We now directly use the $\tilde{m}'$ produced by $\mathcal{A}_{2,j-1}$ instead of $\tilde{m}$. Note that this “new” $\tilde{m}$ is also used to define O'_m , and (implicitly) bad in $\mathcal{G}_4$. Formally, game $\mathcal{G}_4$ is as follows:

$$\begin{array}{l}
\mathcal{G}_4(1^\eta) \\
\rho \leftarrow \$ \\
\hline
j \leftarrow \$ [1, q(\eta)] \\
\tilde{u}, \tilde{v} \leftarrow \mathcal{A}_1^{O_1}(1^\eta, \rho) \\
\rightarrow \tilde{m} \leftarrow \mathcal{A}_{2,j-1}^{O_1}(1^\eta, \rho) \\
\tilde{h} \leftarrow O_1(\tilde{m}) \\
\text{return } (\tilde{v} = \mathcal{A}_4^{O'_m}(1^\eta, \rho, \tilde{u}, \tilde{h})) \wedge \neg \text{bad}
\end{array}$$

The argument here is that, when j is guessed right, $\mathcal{G}_4$ will be just as $\mathcal{G}_3$. By assumption, $\mathcal{A}_2$ always makes during its execution exactly one oracle call on the value it ends up returning.

Formally, let j_0 be the random variable denoting the index of that oracle call, among all those made by $\mathcal{A}_2$, during its execution in $\mathcal{G}_2(1^\eta)$. Equivalently, j_0 is the index of the oracle call on the value produced by $\mathcal{A}_{2,j+1}$ in $\mathcal{G}_3(1^\eta)$, provided calls made by $\mathcal{A}_{2,j-1}$ are numbered 1 to (at most) $j-1$, the call set aside in the game is the j -th, and calls made by $\mathcal{A}_{2,j+1}$ (if any) are numbered starting at $j+1$.

In the execution of $\mathcal{G}_4$, the value selected as $\tilde{m}$ in $\mathcal{G}_4$ is the one produced by $\mathcal{A}_{2,j-1}$ as the j -th oracle call. In $\mathcal{G}_3$, the value selected as $\tilde{m}$ is the one returned by $\mathcal{A}_{2,j+1}$. For any given ρ , if j is guessed equal to j_0 , then these two values are the same, and the two games return the same value. Therefore:

$$\Pr[\mathcal{G}_4(1^\eta)] \geq \Pr[\mathcal{G}_4(1^\eta) \wedge j = j_0] = \Pr[\mathcal{G}_3(1^\eta) \wedge j = j_0].$$

In addition, the event " $j = j_0$ " is independent of the result of $\mathcal{G}_3$, since that is the same as $\mathcal{G}_2$, and j is a uniformly sampled value that plays no role in $\mathcal{G}_2$. Thus

$$\Pr[\mathcal{G}_3(1^\eta) \wedge j = j_0] = \Pr[j = j_0] \cdot \Pr[\mathcal{G}_3(1^\eta)].$$

For the same reason, j is independent of $j_0 - j_0$ is defined from $\mathcal{G}_2$ in which j does not exist, and j is then drawn uniformly at random. Thus, the two are equal with probability $q(\eta)^{-1}$. Therefore:

$$\Pr[\mathcal{G}_4(1^\eta)] \geq q(\eta)^{-1} \cdot \Pr[\mathcal{G}_3(1^\eta)],$$

which is to say

$$\Pr[\mathcal{G}_3(1^\eta)] \leq q(\eta) \cdot \Pr[\mathcal{G}_4(1^\eta)].$$

Fifth hop: We now replace the value $\tilde{h}$ given to $\mathcal{A}_4$ with a random sampling, in $\mathcal{G}_5$:

$$\begin{array}{l} \mathcal{G}_5(1^\eta) \\ \hline \rho \leftarrow \$ \\ j \leftarrow [1, q(\eta)] \\ \tilde{u}, \tilde{v} \leftarrow \mathcal{A}_1^{O_1}(1^\eta, \rho) \\ \neg \neg \tilde{m} \leftarrow \mathcal{A}_{2,j-1}^{O_1}(1^\eta, \rho) \\ \tilde{h} \leftarrow \$ \\ \text{return } (\tilde{v} = \mathcal{A}_4^{O'_m}(1^\eta, \rho, \tilde{u}, \tilde{h})) \wedge \neg \text{bad} \end{array}$$

Note that $\tilde{m}$ still must be computed, since O'_m and bad depend on it.

In $\mathcal{G}_4$, in any execution where $\mathcal{G}_4(1^\eta) = 1$, $\tilde{m}$ is not submitted to O_1 by either $\mathcal{A}_1$ (since that would contradict $\neg \text{bad}_{\mathcal{G}_4}$), $\mathcal{A}_{2,j-1}$ (by construction) or $\mathcal{A}_4$ (O'_m prevents it by construction, and doing so through $\tilde{u}$ would contradict $\neg \text{bad}_{\mathcal{G}_4}$). In these executions, O_1 is thus called on $\tilde{m}$ at exactly one point: when computing $\tilde{h}$. Therefore, identifying the sampling of $\tilde{h}$ in $\mathcal{G}_5$ and the one done internally by $O_1(\tilde{m})$ in $\mathcal{G}_4$, in all corresponding executions of $\mathcal{G}_5$, $\mathcal{G}_5$ does exactly as $\mathcal{G}_4$, and thus returns 1 as well. Thus:

$$\Pr[\mathcal{G}_4(1^\eta)] \leq \Pr[\mathcal{G}_5(1^\eta)].$$

Sixth hop: We now remove the bad event, in $\mathcal{G}_6$:

$$\begin{array}{l} \mathcal{G}_6(1^\eta) \\ \hline \rho \leftarrow \$ \\ j \leftarrow [1, q(\eta)] \\ \tilde{u}, \tilde{v} \leftarrow \mathcal{A}_1^{O_1}(1^\eta, \rho) \\ \neg \neg \tilde{m} \leftarrow \mathcal{A}_{2,j-1}^{O_1}(1^\eta, \rho) \\ \tilde{h} \leftarrow \$ \\ \text{return } (\tilde{v} = \mathcal{A}_4^{O'_m}(1^\eta, \rho, \tilde{u}, \tilde{h})) \end{array}$$

Clearly, $\mathcal{G}_6$ returns 1 at least as often as $\mathcal{G}_5$, and:

$$\Pr[\mathcal{G}_5(1^\eta)] \leq \Pr[\mathcal{G}_6(1^\eta)].$$

Seventh hop: We now reach game $\mathcal{G}_7$, which is rewritten to be closer to an instance of an adversary attempting to contradict $u, o_h^k \not\vdash v$:

$$\begin{array}{l} \mathcal{G}_7(1^\eta) \\ \hline \rho \leftarrow \$ \\ \tilde{u}, \tilde{v} \leftarrow \mathcal{A}_1^{O_1}(1^\eta, \rho) \\ \text{return } (\tilde{v} = \mathcal{A}_5^{O_1}(1^\eta, \rho, \tilde{u})) \end{array}$$

where $\mathcal{A}_5$ is defined as follows:

$$\begin{array}{l} \mathcal{A}_5(1^\eta, \rho, x) \\ \hline \tilde{j} \leftarrow [1, q(\eta)] \\ \neg \neg \tilde{m} \leftarrow \mathcal{A}_{2,j-1}^{O_1}(1^\eta, \rho) \\ \tilde{h} \leftarrow \$ \\ \text{use } O_1 \text{ and } \tilde{m} \text{ to implement } O'_m \text{ in} \\ \text{return } \mathcal{A}_4^{O'_m}(1^\eta, \rho, x, \tilde{h}) \end{array}$$

Going from $\mathcal{G}_6$ to $\mathcal{G}_7$, we have simply reorganised operations, but essentially nothing has changed. Therefore:

$$\Pr[\mathcal{G}_6(1^\eta)] = \Pr[\mathcal{G}_7(1^\eta)].$$

Final hop: Finally, we consider game $\mathcal{G}_8$, which is the same as $\mathcal{G}_7$ but uses O_0 rather than O_1 :

$$\begin{array}{l} \mathcal{G}_8(1^\eta) \\ \hline \rho \leftarrow \$ \\ \tilde{u}, \tilde{v} \leftarrow \mathcal{A}_1^{O_0}(1^\eta, \rho) \\ \text{return } (\tilde{v} = \mathcal{A}_5^{O_0}(1^\eta, \rho, \tilde{u})) \end{array}$$

$(\mathcal{G}_8, \mathcal{G}_7)$ is an instance of the PRF game, and therefore the advantage $\text{Adv}_{7,8}^{\text{prf}} = |\Pr[\mathcal{G}_8] - \Pr[\mathcal{G}_7]|$ is negligible.

In addition, when $\mathcal{A}_1^{O_0}$ correctly computes $\llbracket u \rrbracket_{\mathbb{M}}^{\eta, \rho}, \llbracket v \rrbracket_{\mathbb{M}}^{\eta, \rho}$ - which happens with probability at least $\Pr[\psi_{\text{PRF}, \text{key}}^{k, t}(u, v)]$ - game $\mathcal{G}_8$ is an instance of adversary $\mathcal{A}_5$ attempting to deduce v from u, o_h^k . Note that $\mathcal{A}_5$ only internally calls $\mathcal{A}_4$, and in turn $\mathcal{A}_3$, which indeed only uses the adversarial randomness ρ_a .

Consider the advantage of $\mathcal{A}_5$ in $u, o_h^k \not\vdash v$ (in $\mathbb{M}$) (which, by assumption, is negligible):

$$\text{Adv}_{\mathcal{A}_5}^{u, o_h^k \not\vdash v} \triangleq \Pr_{\rho} \left[\mathcal{A}_5^{O_0}(1^\eta, \rho, \llbracket u \rrbracket_{\mathbb{M}}^{\eta, \rho}) = \llbracket v \rrbracket_{\mathbb{M}}^{\eta, \rho} \right].$$

We then have:

$$\begin{aligned} \Pr [\mathcal{G}_8(1^\eta)] &\leq \Pr \left[\mathcal{G}_8(1^\eta) \wedge \mathcal{A}_1^{O_0}(1^\eta, \rho) = \left(\llbracket u \rrbracket_{\mathbb{M}}^{\eta, \rho}, \llbracket v \rrbracket_{\mathbb{M}}^{\eta, \rho} \right) \right] \\ &\quad + \Pr \left[\mathcal{A}_1^{O_0}(1^\eta, \rho) \neq \left(\llbracket u \rrbracket_{\mathbb{M}}^{\eta, \rho}, \llbracket v \rrbracket_{\mathbb{M}}^{\eta, \rho} \right) \right] \\ &\leq \text{Adv}_{\mathcal{A}_5}^{u, o_h \not\vdash v} + \Pr \left[\neg \llbracket \psi_{\text{PRF, key}}^{k, t}(u, v) \rrbracket_{\mathbb{M}}^{\eta, \rho} \right] \end{aligned}$$

And therefore:

$$\Pr [\mathcal{G}_7(1^\eta)] \leq \text{Adv}_{\mathcal{A}_5}^{u, o_h \not\vdash v} + \text{Adv}_{7,8}^{\text{prf}} + \Pr \left[\neg \llbracket \psi_{\text{PRF, key}}^{k, t}(u, v) \rrbracket_{\mathbb{M}}^{\eta, \rho} \right].$$

Altogether, we have proved that

$$\begin{aligned} \Pr_\rho \left[\text{Exp}_{\mathcal{B}}^{\text{prf}, 1}(1^\eta, \rho) = 1 \right] &\leq p(\eta) \cdot q(\eta) \cdot \left(\text{Adv}_{\mathcal{A}_5}^{u, o_h \not\vdash v} + \text{Adv}_{7,8}^{\text{prf}} + \Pr \left[\neg \llbracket \psi_{\text{PRF, key}}^{k, t}(u, v) \rrbracket_{\mathbb{M}}^{\eta, \rho} \right] \right) \\ &\quad + \Pr_\rho \left[\neg \llbracket \psi_{\text{PRF, msg}}^{m, k}(u, v) \rrbracket_{\mathbb{M}}^{\eta, \rho} \right] \\ &\quad + \text{Adv}_0^{\text{prf}}. \end{aligned}$$

Therefore, $\Pr_\rho \left[\text{Exp}_{\mathcal{B}}^{\text{prf}, 1}(1^\eta, \rho) = 1 \right] \in \text{negl}(\eta)$, which concludes the proof. $\square$

REMARK 1. With rule **PRF.L-EQ**, we consider an additional premise to the rule, stating that $u, o_h \triangleright \lambda x.x = m$, then that oracle can be used in the hop from game $\mathcal{G}_1$ to game $\mathcal{G}_2$, to avoid having to guess which oracle call is on $\tilde{m}$, which should get rid of the $p(\eta)$ factor above.

Alternatively, in rule **PRF.L-SEC**, we use the premise $u, o_h \not\vdash m$ to prove that $\mathcal{A}_3$ is not able to send $\tilde{m}$ to the oracle. The factor $p(\eta)$ also disappears.

Finally, if $\psi_{\text{PRF, msg}}^{m, k}(m)$ also holds, i.e. m is not hashed in m , much of the proof, which was there to obtain the oracle call $\mathcal{A}_2$ makes on $\tilde{m}$, is simplified: j no longer has to be guessed, and the $q(\eta)$ factor should disappear as well.

C.3.2 Right rule.

PRF.R

$$\frac{\begin{array}{c} t \text{ has type } \tau \quad \tau \text{ is serializable} \\ \mathcal{E}; \Theta \vdash \text{adv}(t) \quad \mathcal{E}; \Theta; \emptyset \vdash_{\text{pptm}} u, v \quad \mathcal{E}; \Theta \vdash u, o_h^k \not\vdash v \\ \mathcal{E}; \Theta \vdash \llbracket \psi_{\text{PRF, key}}^{k, t}(u, v) \rrbracket_{\mathbb{M}}^{\eta, \rho} \quad \mathcal{E}; \Theta \vdash \llbracket \psi_{\text{PRF, msg}}^{v, k}(u) \rrbracket_{\mathbb{M}}^{\eta, \rho} \end{array}}{\mathcal{E}; \Theta \vdash u, o_h^k \not\vdash h(v, k(t))}$$

Same as for the left rule, $\psi_{\text{PRF, msg}}^{v, k}(v)$ is not required, τ needs to be encodable as polynomial-length bitstrings, and no syntactic conditions are checked on the oracle o_h^k .

PROOF. By contradiction, assume the rule is unsound. Consider a model $\mathbb{M}$ for which the premises of the rules are satisfied, and such that $\mathbb{M} \models \Theta$, but where $\tilde{v}f. \text{adv}(f) \not\Rightarrow \llbracket f(u, o_h^k) \neq h(v, k(t)) \rrbracket$ does not hold.

Thus, there exists an adversary $\mathcal{A}_3 \in \text{PPTMs}$ that computes the hash from u and the oracle o_h :

$$\Pr \left[\mathcal{A}_3^{O_h}(1^\eta, \rho_a, \llbracket u \rrbracket_{\mathbb{M}}^{\eta, \rho}) = \llbracket h(v, k(t)) \rrbracket_{\mathbb{M}}^{\eta, \rho} \right] \notin \text{negl}(\eta). \quad (6)$$

As in the left case, we call this adversary $\mathcal{A}_3$ for consistency later in the proof, write $\mathcal{A}_3^{O_h}$ to specify $\mathcal{A}_3$ accesses function $\llbracket o_h^k \rrbracket_{\mathbb{M}}^{\eta, \rho}$ as an oracle, and usually write $\mathcal{A}_3(\_, \rho, \_)$ rather than ρ_a for legibility.

Consider the PRF game:

$$\begin{array}{c} \text{Exp}_{\mathcal{B}}^{\text{prf}, \beta}(1^\eta, \rho) \\ \hline k_0 \leftarrow \rho \\ \text{return } \llbracket h \rrbracket_{\mathbb{M}}^{\eta, \rho}(1^\eta, w, k_0) \\ \text{L} \leftarrow \perp \\ \beta' \leftarrow \mathcal{B}^{O_\beta}(1^\eta, \rho) \\ \text{return } \beta' \end{array} \quad \begin{array}{c} O_0(w) \\ \hline \text{return } \llbracket h \rrbracket_{\mathbb{M}}^{\eta, \rho}(1^\eta, w, k_0) \\ \text{if } \text{L}[w] = \perp \text{ then} \\ \text{L}[w] \leftarrow \{0, 1\}^\eta \\ \text{return } \text{L}[w] \end{array}$$

The advantage of $\mathcal{B}$, who can access all the tape ρ except k_0 , is

$$\text{Adv}_{\mathcal{B}}^{\text{prf}}(\eta) = \left| \Pr_\rho \left[\text{Exp}_{\mathcal{B}}^{\text{prf}, 0}(1^\eta, \rho) = 1 \right] - \Pr_\rho \left[\text{Exp}_{\mathcal{B}}^{\text{prf}, 1}(1^\eta, \rho) = 1 \right] \right|.$$

The PRF assumption states that any PPTM $\mathcal{B}$ has a negligible advantage. By the same argument as in the left case, $\llbracket t \rrbracket_{\mathbb{M}}^{\eta, \rho}$ is an adversarial bitstring, and we then consider the instance of the PRF game where k_0 is the sampling $\llbracket k(t) \rrbracket_{\mathbb{M}}^{\eta, \rho}$.

We will use $\mathcal{A}_3$ to construct an adversary $\mathcal{B}$ that wins the PRF game.

High-level intuition. Roughly, $\mathcal{B}$ proceeds as follows. First, it attempts to compute u and v , and $h(v, k(t))$, using oracle O_β as needed to compute hashes with $k(t)$. Then, it calls $\mathcal{A}_3$ on u and o_h , which is simulated by forwarding all queries from $\mathcal{A}_3$ to the oracle O_β provided by the PRF game. Finally, $\mathcal{B}$ checks whether $\mathcal{A}_3$ returned $h(v, k(t))$. If so, it answers $\beta' = 1$, and otherwise $\beta' = 0$.

If $\mathcal{B}$ uses the real hashing oracle O_0 , then it correctly simulates $\mathcal{A}_3^{O_h}(1^\eta, \rho_a, \llbracket u \rrbracket_{\mathbb{M}}^{\eta, \rho})$ and $\llbracket h(v, k(t)) \rrbracket_{\mathbb{M}}^{\eta, \rho}$, and thus answers 1 with the same probability as $\mathcal{A}_3$'s in (5), which is non-negligible. However, if $\mathcal{B}$ uses the idealised oracle O_1 , the hash $\mathcal{B}$ compares to $\mathcal{A}_3$'s output is essentially a random value, chosen (depending on v) among those produced by the oracle O_1 . We will show that, since it cannot compute v from u and O_1 (by assumption), $\mathcal{A}_3$ has at most a negligible probability of finding that random value. This will establish that, when $\beta = 1$, $\mathcal{B}$ only answers 1 with negligible probability, and thus that $\mathcal{B}$ wins the PRF game.

Defining $\mathcal{B}$. Let us now define $\mathcal{B}$ formally. Since $\mathcal{E}; \Theta; \emptyset \vdash_{\text{pptm}} u, v$ and $\mathcal{E}; \Theta; \emptyset \vdash \psi_{\text{PRF, key}}^{k, t}(u, v)$, there exists a PPTM $\mathcal{A}_1^{O_0}(1^\eta, \rho)$ that computes $\llbracket u \rrbracket_{\mathbb{M}}^{\eta, \rho}$, and in doing so never needs to access the part of ρ associated with $k(t)$, and calls the hashing oracle O_0 when a hash with that key is needed. Similarly, there exists a PPTM $\mathcal{A}_2^{O_0}(1^\eta, \rho)$, that computes $\llbracket v \rrbracket_{\mathbb{M}}^{\eta, \rho}$.

We have:

$$\begin{aligned} \Pr_\rho \left[\mathcal{A}_1^{O_0}(\rho) = \llbracket u \rrbracket_{\mathbb{M}}^{\eta, \rho} \wedge \mathcal{A}_2^{O_0}(\rho) = \llbracket v \rrbracket_{\mathbb{M}}^{\eta, \rho} \right] \\ \geq \Pr_\rho \left[\llbracket \psi_{\text{PRF, key}}^{k, t}(u, v) \rrbracket_{\mathbb{M}}^{\eta, \rho} \right] \in \text{ow}(\eta). \end{aligned}$$

These machine proceed as in the left case, recursively evaluating the terms, and calling the oracle whenever a hash with $k(t)$ is needed.

The adversary $\mathcal{B}$ is then defined as follows:

$$\begin{array}{c} \mathcal{B}^{O_\beta}(1^\eta, \rho) \\ \hline \tilde{u} \leftarrow \mathcal{A}_1^{O_\beta}(1^\eta, \rho) \\ \tilde{v} \leftarrow \mathcal{A}_2^{O_\beta}(1^\eta, \rho) \\ \tilde{h} \leftarrow O_\beta(\tilde{v}) \\ \text{return } (\tilde{h} = \mathcal{A}_3^{O_\beta}(1^\eta, \rho, \tilde{u})) \end{array}$$

$\mathcal{B}$ executes machines $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$, using its oracle O_β to answer hash oracle calls, and the tape ρ to provide all randomness except the hashing key – which the three machines never ask for by construction.

As before, we assume w.l.o.g. that each one of $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$ never calls the oracle on the same value twice, and that $\mathcal{A}_2$, on any input, always makes during its execution a call to its oracle on the value $\tilde{v}$ it returns.

Proving that $\mathcal{B}$ wins against PRF. If $\beta = 0$, the simulators $\mathcal{A}_1^{O_0}, \mathcal{A}_2^{O_0}$ are run as expected, and the oracle O_0 given to $\mathcal{A}_3$ behaves exactly as o_h . Thus

$$\left| \Pr_\rho \left[\text{Exp}_{\mathcal{B}}^{\text{prf},0}(1^\eta, \rho) = 1 \right] - \Pr_\rho \left[\mathcal{A}_3^{O_h}(1^\eta, \rho, \llbracket u \rrbracket_{\mathbb{M}}^{\eta,\rho}) = \llbracket h(v, k(t)) \rrbracket_{\mathbb{M}}^{\eta,\rho} \right] \right| \in \text{negl}(\eta)$$

which means, since $\mathcal{A}_3$ has a non-negligible advantage, that

$$\Pr_\rho \left[\text{Exp}_{\mathcal{B}}^{\text{prf},0}(1^\eta, \rho) = 1 \right] \notin \text{negl}(\eta).$$

Consider now the case $\beta = 1$. We will now show, by a series of game hops, that

$$\Pr_\rho \left[\text{Exp}_{\mathcal{B}}^{\text{prf},1}(1^\eta, \rho) = 1 \right] \in \text{negl}(\eta),$$

which naturally implies, together with the previous case, that $\mathcal{B}$ wins the PRF game with a non-negligible advantage.

Initial game: We consider the following game $\mathcal{G}_0$:

$$\begin{aligned} & \frac{\mathcal{G}_0(1^\eta)}{\rho \leftarrow \$} \\ & \tilde{u} \leftarrow \mathcal{A}_1^{O_1}(1^\eta, \rho) \\ & \tilde{v} \leftarrow \mathcal{A}_2^{O_1}(1^\eta, \rho) \\ & \tilde{h} \leftarrow O_1(\tilde{v}) \\ & \text{return } \tilde{h} = \mathcal{A}_3^{O_1}(1^\eta, \rho, \tilde{u}) \end{aligned}$$

$\mathcal{G}_0$ is in fact the right side of the PRF game with $\mathcal{B}$:

$$\Pr_\rho \left[\text{Exp}_{\mathcal{B}}^{\text{prf},1}(1^\eta, \rho) = 1 \right] = \Pr \left[\mathcal{G}_0(1^\eta, \rho) \right].$$

First hop: We first exclude the possibility of $\mathcal{A}_1$ calling the oracle O_1 on $\tilde{v}$. We consider the game $\mathcal{G}_1$:

$$\begin{aligned} & \frac{\mathcal{G}_1(1^\eta)}{\rho \leftarrow \$} \\ & \tilde{u} \leftarrow \mathcal{A}_1^{O_1}(1^\eta, \rho) \\ & \tilde{v} \leftarrow \mathcal{A}_2^{O_1}(1^\eta, \rho) \\ & \tilde{h} \leftarrow O_1(\tilde{v}) \\ & \text{return } (\tilde{h} = \mathcal{A}_3^{O_1}(1^\eta, \rho, \tilde{u})) \wedge \neg \text{bad} \end{aligned}$$

where bad is defined as follows. Let log denote the log of all values on which O_1 is queried by $\mathcal{A}_1$ during the experiment, either during

its execution, or to evaluate first-order functions returned as closures as part of $\tilde{u}$, that $\mathcal{A}_3$ may later call during its own execution. bad is then defined as the condition $\tilde{v} \in \text{log}$. bad is tied to a given experiment, here $\mathcal{G}_1$ – e.g., if $\mathcal{A}_2$ or $\mathcal{A}_3$ were called on different parameters, then the value of $\tilde{v}$ and the oracle's log could differ, and thus so could bad. As before, we may write $\text{bad}_{\mathcal{G}}$ to specify unambiguously that $\tilde{v}$ and log are to be understood w.r.t. the execution of a game $\mathcal{G}$. Note for instance that $\text{bad}_{\mathcal{G}_0} = \text{bad}_{\mathcal{G}_1}$.

We have

$$\Pr \left[\mathcal{G}_0(1^\eta) \right] \leq \Pr \left[\mathcal{G}_1(1^\eta) \right] + \Pr \left[\text{bad}_{\mathcal{G}_0} \right].$$

Besides, consider the game $\mathcal{G}_0^\beta$ for $\beta \in \{0, 1\}$:

$$\begin{aligned} & \frac{\mathcal{G}_0^\beta(1^\eta)}{\rho \leftarrow \$} \\ & \tilde{u} \leftarrow \mathcal{A}_1^{O_\beta}(1^\eta, \rho) \\ & \tilde{v} \leftarrow \mathcal{A}_2^{O_\beta}(1^\eta, \rho) \\ & \tilde{h} \leftarrow O_\beta(\tilde{v}) \\ & \_ \leftarrow \mathcal{A}_3^{O_\beta}(1^\eta, \rho, \tilde{u}) \\ & \text{return bad} \end{aligned}$$

$(\mathcal{G}_0^0, \mathcal{G}_0^1)$ is itself an instance of the PRF game, and therefore the associated advantage $\text{Adv}_0^{\text{prf}} = \left| \Pr \left[\text{bad}_{\mathcal{G}_0^1} \right] - \Pr \left[\text{bad}_{\mathcal{G}_0^0} \right] \right|$ is negligible.

In addition, we know, by construction of $\psi_{\text{PRF,msg}}^{v,k}(u)$, that

$$\Pr \left[\text{bad}_{\mathcal{G}_0^0} \right] \leq \Pr_\rho \left[\neg \llbracket \psi_{\text{PRF,msg}}^{v,k}(u) \rrbracket_{\mathbb{M}}^{\eta,\rho} \right].$$

Since $\text{bad}_{\mathcal{G}_0^1}$ is in fact the same as $\text{bad}_{\mathcal{G}_0}$, we get that:

$$\Pr \left[\text{bad}_{\mathcal{G}_0} \right] \leq \Pr_\rho \left[\neg \llbracket \psi_{\text{PRF,msg}}^{v,k}(u) \rrbracket_{\mathbb{M}}^{\eta,\rho} \right] + \text{Adv}_0^{\text{prf}}.$$

Therefore:

$$\Pr \left[\mathcal{G}_0(1^\eta) \right] \leq \Pr \left[\mathcal{G}_1(1^\eta) \right] + \Pr_\rho \left[\neg \llbracket \psi_{\text{PRF,msg}}^{v,k}(u) \rrbracket_{\mathbb{M}}^{\eta,\rho} \right] + \text{Adv}_0^{\text{prf}}.$$

Second hop: We now remove the possibility for $\mathcal{A}_3$ to call the oracle on $\tilde{v}$, which is not an issue, since $\mathcal{A}_3$ is supposed to be unable to compute $\tilde{v}$ anyway.

For any message x , we let O'_x be the oracle O_1 punctured on input x :

$$O'_x(y) = \begin{cases} O_1(y) & \text{if } y \neq x \\ \perp & \text{otherwise} \end{cases}$$

We consider the game $\mathcal{G}_2$:

$$\begin{aligned} & \frac{\mathcal{G}_2(1^\eta)}{\rho \leftarrow \$} \\ & \tilde{u} \leftarrow \mathcal{A}_1^{O_1}(1^\eta, \rho) \\ & \tilde{v} \leftarrow \mathcal{A}_2^{O_1}(1^\eta, \rho) \\ & \tilde{h} \leftarrow O_1(\tilde{v}) \\ & \text{return } (\tilde{h} = \mathcal{A}_3^{O'_v}(1^\eta, \rho, \tilde{u})) \wedge \neg \text{bad} \end{aligned}$$

The executions of $\mathcal{G}_1$ and $\mathcal{G}_2$ may only differ when $\mathcal{A}_3$ (attempts to) make a call to its oracle on $\tilde{v}$. Let call denote the event “ $\mathcal{A}_3$ calls its oracle on $\tilde{v}$ in $\mathcal{G}_1$ ”. We therefore have:

$$\left| \Pr [\mathcal{G}_1(1^\eta)] - \Pr [\mathcal{G}_2(1^\eta)] \right| \leq \Pr [\text{call}]$$

which implies that

$$\Pr [\mathcal{G}_1(1^\eta)] \leq \Pr [\mathcal{G}_2(1^\eta)] + \Pr [\text{call}].$$

We will now bound $\Pr [\text{call}]$. As before, we construct $\mathcal{A}_4$ that guesses one oracle input from $\mathcal{A}_3$. By the same argument as for the left rule, there exists a polynomial $p(\eta)$ that strictly bounds the number of calls made by $\mathcal{A}_3$ to its oracle in any execution of $\mathcal{G}_1(1^\eta)$. Let then $\mathcal{A}_4$ be the following machine:

$\mathcal{A}_4^O(1^\eta, \rho, x)$
 $\tilde{i} \leftarrow \llbracket 1, p(\eta) \rrbracket$
 run $\mathcal{A}_3^O(1^\eta, \rho, x)$ normally, until its
 i -th call to O : do not make that call,
 and return the value input to O .
 If no i -th call is made, return $\perp$.

Consider now the following pair of games $\mathcal{G}_1^\beta$ for $\beta \in \{0, 1\}$:

$\mathcal{G}_1^\beta(1^\eta)$
 $\rho \leftarrow \mathbb{S}$
 $\tilde{u} \leftarrow \mathcal{A}_1^{O_\beta}(1^\eta, \rho)$
 $\tilde{v} \leftarrow \mathcal{A}_2^{O_\beta}(1^\eta, \rho)$
return $(\tilde{v} = \mathcal{A}_4^{O_\beta}(1^\eta, \rho, \tilde{u}))$

$(\mathcal{G}_1^0, \mathcal{G}_1^1)$ forms an instance of the PRF game, and therefore the associated advantage $\text{Adv}_1^{\text{prf}} \triangleq \left| \Pr [\mathcal{G}_1^0(1^\eta)] - \Pr [\mathcal{G}_1^1(1^\eta)] \right|$ is negligible.

In addition, when $\mathcal{A}_1, \mathcal{A}_2$ correctly compute $\llbracket u \rrbracket_{\mathbb{M}}^{\eta, \rho}, \llbracket v \rrbracket_{\mathbb{M}}^{\eta, \rho}$ – which happens with probability at least $\Pr [\psi_{\text{PRF, key}}^{k, t}(u, v)]$ – game $\mathcal{G}_1^0$ is an instance of adversary $\mathcal{A}_4$ attempting to deduce v from u, o_h^k . Note that $\mathcal{A}_4$ only internally calls $\mathcal{A}_3$, which only uses the adversarial randomness ρ_a . Consider the advantage of $\mathcal{A}_4$ in $u, o_h^k \not\equiv v$ (in $\mathbb{M}$) (which, by assumption, is negligible):

$$\text{Adv}_{\mathcal{A}_4}^{u, o_h^k \not\equiv v} \triangleq \Pr_{\rho} \left[\mathcal{A}_4^{O_0}(1^\eta, \rho, \llbracket u \rrbracket_{\mathbb{M}}^{\eta, \rho}) = \llbracket v \rrbracket_{\mathbb{M}}^{\eta, \rho} \right].$$

We then have:

$$\begin{aligned} \Pr [\mathcal{G}_1^0(1^\eta)] &\leq \Pr \left[\mathcal{G}_1^0 \wedge \mathcal{A}_1^{O_0}(1^\eta, \rho) = \llbracket u \rrbracket_{\mathbb{M}}^{\eta, \rho} \wedge \mathcal{A}_2^{O_0}(1^\eta, \rho) = \llbracket v \rrbracket_{\mathbb{M}}^{\eta, \rho} \right] \\ &\quad + \Pr \left[\mathcal{A}_1^{O_0}(1^\eta, \rho) \neq \llbracket u \rrbracket_{\mathbb{M}}^{\eta, \rho} \vee \mathcal{A}_2^{O_0}(1^\eta, \rho) \neq \llbracket v \rrbracket_{\mathbb{M}}^{\eta, \rho} \right] \\ &\leq \text{Adv}_{\mathcal{A}_4}^{u, o_h^k \not\equiv v} + \Pr \left[\neg \llbracket \psi_{\text{PRF, key}}^{k, t}(u, v) \rrbracket_{\mathbb{M}}^{\eta, \rho} \right] \end{aligned}$$

And therefore:

$$\Pr [\mathcal{G}_1^1(1^\eta)] \leq \text{Adv}_1^{\text{prf}} + \text{Adv}_{\mathcal{A}_4}^{u, o_h^k \not\equiv v} + \Pr \left[\neg \llbracket \psi_{\text{PRF, key}}^{k, t}(u, v) \rrbracket_{\mathbb{M}}^{\eta, \rho} \right].$$

Moreover, consider the execution of $\mathcal{A}_3$ in $\mathcal{G}_1$. Let then i_0 be the random variable defined as the index of $\mathcal{A}_3$'s oracle call on $\tilde{v}$, if there is one, and $p(\eta)$ otherwise. For any given ρ , if $\mathcal{A}_3$ does make a call to O_1 on $\tilde{v}$, and if i is guessed such that $i = i_0$, then $\mathcal{G}_1^1$ returns 1. Therefore:

$$\Pr [\mathcal{G}_1^1(1^\eta)] \geq \Pr [\text{call} \wedge i = i_0].$$

The event “ $i = i_0$ ” is independent of the execution of $\mathcal{G}_1$, and thus from call, since i is a fresh uniformly sampled value that does not appear in $\mathcal{G}_1$. For the same reason, $\Pr [i = i_0] = p(\eta)^{-1}$. Thus:

$$\Pr [\mathcal{G}_1^1(1^\eta)] \geq \Pr [\text{call}] \cdot \Pr [i = i_0] = p(\eta)^{-1} \cdot \Pr [\text{call}].$$

Altogether, we get:

$$\begin{aligned} \Pr [\mathcal{G}_1(1^\eta)] &\leq \Pr [\mathcal{G}_2(1^\eta)] \\ &\quad + p(\eta) \cdot \left(\text{Adv}_1^{\text{prf}} + \text{Adv}_{\mathcal{A}_4}^{u, o_h^k \not\equiv v} + \Pr \left[\neg \llbracket \psi_{\text{PRF, key}}^{k, t}(u, v) \rrbracket_{\mathbb{M}}^{\eta, \rho} \right] \right). \end{aligned}$$

Third hop: Next, we decompose $\mathcal{A}_2$'s execution, to isolate its call to O_1 on $\tilde{v}$ (which exists and is unique by assumption). As before, since $\mathcal{A}_2$ does not know *a priori* the value of $\tilde{v}$, we have to guess which oracle call is the right one.

Again, $\mathcal{A}_2$ is a PPTM, and thus there exists a polynomial $q(\eta)$ that bounds the number of calls $\mathcal{A}_2^{O_1}(1^\eta, \rho)$ makes to O_1 , for all ρ . We introduce the following game $\mathcal{G}_3$:

$\mathcal{G}_3(1^\eta)$
 $\rho \leftarrow \mathbb{S}$
 $j \leftarrow \llbracket 1, q(\eta) \rrbracket$
 $\tilde{u} \leftarrow \mathcal{A}_1^{O_1}(1^\eta, \rho)$
 done, st, $\tilde{v}' \leftarrow \mathcal{A}_{2, j-1}^{O_1}(1^\eta, \rho)$
 $\tilde{h}' \leftarrow \text{if } \neg \text{done} \text{ then } O_1(\tilde{v}') \text{ else } \perp$
 $\tilde{v} \leftarrow \mathcal{A}_{2, j+1}^{O_1}(1^\eta, \rho, \text{done, st, } \tilde{v}', \tilde{h}')$
 $\tilde{h} \leftarrow O_1(\tilde{v})$
return $(\tilde{h} = \mathcal{A}_3^{O'_v}(1^\eta, \rho, \tilde{u})) \wedge \neg \text{bad}$

where $\mathcal{A}_{2, j-1}, \mathcal{A}_{2, j+1}$ represent the execution of $\mathcal{A}_2$ before its j -th call to O_1 , and after, as in the previous proof. No matter the value drawn for j , the execution of $\mathcal{A}_2$ simulated by $\mathcal{A}_{2, j-1}$ and $\mathcal{A}_{2, j+1}$ is accurate, and therefore:

$$\Pr [\mathcal{G}_2(1^\eta)] = \Pr [\mathcal{G}_3(1^\eta)].$$

Fourth hop: We now directly use the $\tilde{v}'$ produced by $\mathcal{A}_{2, j-1}$ in stead of $\tilde{v}$. Formally, game $\mathcal{G}_4$ is as follows:

$\mathcal{G}_4(1^\eta)$
 $\rho \leftarrow \mathbb{S}$
 $j \leftarrow \llbracket 1, q(\eta) \rrbracket$
 $\tilde{u} \leftarrow \mathcal{A}_1^{O_1}(1^\eta, \rho)$
 $\neg \neg, \tilde{v} \leftarrow \mathcal{A}_{2, j-1}^{O_1}(1^\eta, \rho)$
 $\tilde{h} \leftarrow O_1(\tilde{v})$
return $(\tilde{h} = \mathcal{A}_3^{O'_v}(1^\eta, \rho, \tilde{u})) \wedge \neg \text{bad}$

SK:	Initial shared secret
(sk_i^I, pk_i^I) :	Initiator fresh DH key-pair at step i
(sk_i^R, pk_i^R) :	Responder fresh DH key-pair at step i
$RK_{i,s}^I$:	Initiator sending root key at step i
$RK_{i,r}^R$:	Responder receiving root key at step i (equal to $RK_{i,s}^{I-1}$ when synced)
$RK_{i,s}^R$:	Responder sending root key at step i
$RK_{i,r}^I$:	Initiator receiving root key at step i (equal to $RK_{i,r}^R$ when synced)
$CK_{i,s}, CK_{i,r}, CK_{R,s}, \dots$:	Chain keys derived at the same ratchet step as its corresponding RK

Table 2: Summary of all DR values.

Note that the “new” $\tilde{v}$ is also used to define O'_v , and (implicitly) bad in $\mathcal{G}_4$. The argument here is that, when j is guessed right, $\mathcal{G}_4$ will be just as $\mathcal{G}_3$. Following the exact same reasoning as in the left-rule proof, we have:

$$\Pr [\mathcal{G}_3(1^\eta)] \leq q(\eta) \cdot \Pr [\mathcal{G}_4(1^\eta)].$$

Fifth hop: We now replace the value $\tilde{h}$ with a random sampling, in $\mathcal{G}_5$:

$$\begin{array}{l}
\mathcal{G}_5(1^\eta) \\
\rho \leftarrow \$ \\
j \leftarrow [1, q(\eta)] \\
\tilde{u} \leftarrow \mathcal{A}_1^{O_1}(1^\eta, \rho) \\
\rightarrow, \tilde{v} \leftarrow \mathcal{A}_{2,j-1}^{O_1}(1^\eta, \rho) \\
\tilde{h} \leftarrow \$ \\
\text{return } (\tilde{h} = \mathcal{A}_3^{O'_v}(1^\eta, \rho, \tilde{u})) \wedge \neg \text{bad}
\end{array}$$

Following the same reasoning as in the previous proof, in any execution where $\mathcal{G}_4(1^\eta) = 1$, $\tilde{v}$ is not submitted to O_1 by either $\mathcal{A}_1$ (that would contradict $\neg \text{bad}_{\mathcal{G}_4}$), $\mathcal{A}_{2,j-1}$ (by construction) or $\mathcal{A}_3$ (O'_v prevents it by construction, and doing so through $\tilde{u}$ would contradict $\neg \text{bad}_{\mathcal{G}_4}$). In these executions, $\tilde{h}$ is thus indeed a random sampling independently drawn, and:

$$\Pr [\mathcal{G}_4(1^\eta)] \leq \Pr [\mathcal{G}_5(1^\eta)].$$

Sixth hop: We now remove the bad event, in $\mathcal{G}_6$:

$$\begin{array}{l}
\mathcal{G}_6(1^\eta) \\
\rho \leftarrow \$ \\
j \leftarrow [1, q(\eta)] \\
\tilde{u} \leftarrow \mathcal{A}_1^{O_1}(1^\eta, \rho) \\
\rightarrow, \tilde{v} \leftarrow \mathcal{A}_{2,j-1}^{O_1}(1^\eta, \rho) \\
\tilde{h} \leftarrow \$ \\
\text{return } (\tilde{h} = \mathcal{A}_3^{O'_v}(1^\eta, \rho, \tilde{u}))
\end{array}$$

Clearly, $\mathcal{G}_6$ returns 1 at least as often as $\mathcal{G}_5$, and:

$$\Pr [\mathcal{G}_5(1^\eta)] \leq \Pr [\mathcal{G}_6(1^\eta)].$$

Final hop: We now reach game $\mathcal{G}_7$:

$$\begin{array}{l}
\mathcal{G}_7(1^\eta) \\
\rho \leftarrow \$ \\
\tilde{h} \leftarrow \$ \\
\text{return } (\tilde{h} = \mathcal{A}_5^{O_1}(1^\eta, \rho))
\end{array}$$

where $\mathcal{A}_5$ is defined as follows:

$$\begin{array}{l}
\mathcal{A}_5(1^\eta, \rho) \\
j \leftarrow [1, q(\eta)] \\
\tilde{u} \leftarrow \mathcal{A}_1^{O_1}(1^\eta, \rho) \\
\rightarrow, \tilde{v} \leftarrow \mathcal{A}_{2,j-1}^{O_1}(1^\eta, \rho) \\
\text{use } O_1 \text{ and } \tilde{v} \text{ to implement } O'_v \text{ in} \\
\text{return } \mathcal{A}_3^{O'_v}(1^\eta, \rho, \tilde{u})
\end{array}$$

Going from $\mathcal{G}_6$ to $\mathcal{G}_7$, we have simply reorganised operations, but essentially nothing has changed. Therefore:

$$\Pr [\mathcal{G}_6(1^\eta)] = \Pr [\mathcal{G}_7(1^\eta)].$$

Game $\mathcal{G}_7$ is an instance of a machine $\mathcal{A}_5$, with access to a random oracle O_1 , attempting to guess the value of an independently, uniformly sampled random bitstring $\tilde{h}$ of length η . Such an experiment has a probability of success of $2^{-\eta}$:

$$\Pr [\mathcal{G}_7(1^\eta)] = 2^{-\eta}.$$

Altogether, we have proved that

$$\begin{aligned}
& \Pr_\rho \left[\text{Exp}_{\mathcal{B}}^{\text{prf},1}(1^\eta, \rho) = 1 \right] \\
& \leq p(\eta) \cdot \left(\text{Adv}_{\mathcal{A}_4}^{u, o_h, \tilde{v}} + \text{Adv}_1^{\text{prf}} + \Pr \left[-\|\psi_{\text{PRF}, \text{key}}^{k,t}(u, v)\|_{\mathcal{M}}^{\eta, \rho} \right] \right) \\
& + q(\eta) \cdot 2^{-\eta}.
\end{aligned}$$

Therefore, $\Pr_\rho \left[\text{Exp}_{\mathcal{B}}^{\text{prf},1}(1^\eta, \rho) = 1 \right] \in \text{negl}(\eta)$, which concludes the proof. $\square$

D Additional Details on the Ratchet Analysis

We provide some missing details on the security analysis of the asymmetric ratchet. In Table 2, we summarize the notations used throughout Section 4. In Fig. 13, we present our set of oracles O .

E Issues in Previous Pen-and-Paper Proofs

Below, we discuss the seminal pen-and-paper proof of the DR from [25]. Consider the setting depicted in [25, Fig. 10]. There is a subtlety, which is that neither rk_1 nor $ck_{0,1}^{ir}$ are RoR, because both rk_2 and $ck_{0,2}^{ir}$ could be leaked: they are painted in gray in the figure, but should be in red. Indeed, as we pointed out in Section 1, PCS allows later compromise of the chains, which break the RoR secrecy of past keys.

This leads to a subtle situation, where the keys are only non-deducible by an attacker and not RoR. This has a small impact on the proof in [25], and makes several statements and arguments incorrect. Nevertheless, we believe that the development in [25] could be modified to fix this issue with only minor changes, though we did not check this in depth. In particular, the situation is not as

```

let  $O_{\text{ddh}}^X = \lambda y \times i. y = x^{\text{sk}_X^i}$  (* DDH oracle where  $X \in \{I, R\}$  *)
let  $O_{\text{leak}}^R = \lambda i. \text{if not (safe}_X(i)) \text{ then sk}_X^i$  (* leaked DH secrets where  $X \in \{I, R\}$  *)
let  $O_{\text{public}} =$ 
  ( $O_{\text{ddh}}^R, O_{\text{ddh}}^I, O_{\text{leak}}^R, O_{\text{leak}}^I,$ 
    $\lambda x. h(x, k), \lambda x. h'(x, k'),$ 
    $\lambda i. g^{\text{sk}_R^i}, \lambda i. g^{\text{sk}_I^i},$ 
    $h(RK_{I,s}^0, k')$ )
(* random oracles *)
(* public DH keys *)
(* ad hoc case for initial half-ratchet *)

```

```

(* Healthy sending chain keys. *)
let  $O_{\text{CK}}^{\text{R},s}(\tau) = \lambda i. \text{if } R(i) \leq \tau \wedge \text{healthy}_R(i) \text{ then CK}_{R,s}^i.$ 
let  $O_{\text{CK}}^{\text{I},s}(\tau) = \lambda i. \text{if } I(i) \leq \tau \wedge \text{healthy}_I(i) \text{ then CK}_{I,s}^i.$ 

(* Healthy de-synchronized receiving chain keys. *)
let  $O_{\text{CK}}^{\text{R},r}(\tau) = \lambda i. \text{if } R(i) \leq \tau \wedge \text{healthy}_R(i-1) \wedge \text{not(sync}_{R(I)}(i-1)) \text{ then CK}_{R,r}^i.$ 
let  $O_{\text{CK}}^{\text{I},r}(\tau) = \lambda i. \text{if } I(i) \leq \tau \wedge \text{healthy}_I(i-1) \wedge \text{not(sync}_{R(I)}(i)) \text{ then CK}_{I,r}^i.$ 

(* Non-healthy sending root keys. *)
let  $O_{\text{RK}}^{\text{R},s}(\tau) = \lambda i. \text{if } R(i) \leq \tau \wedge \text{not(healthy}_R(i)) \text{ then RK}_{R,s}^i.$ 
let  $O_{\text{RK}}^{\text{I},s}(\tau) = \lambda i. \text{if } I(i) \leq \tau \wedge \text{not(healthy}_I(i)) \text{ then RK}_{I,s}^i.$ 

(* Non-healthy de-synchronized receiving root keys. *)
let  $O_{\text{RK}}^{\text{R},r}(\tau) = \lambda i. \text{if } R(i) \leq \tau \wedge \text{not(sync}_{R(I)}(i-1)) \wedge \text{not(healthy}_R(i-1)) \text{ then RK}_{R,r}^i.$ 
let  $O_{\text{RK}}^{\text{I},r}(\tau) = \lambda i. \text{if } I(i) \leq \tau \wedge \text{not(sync}_{R(I)}(i)) \wedge \text{not(healthy}_I(i-1)) \text{ then RK}_{I,r}^i.$ 

(* Putting all oracles together. *)
let  $O(\tau) = (O_{\text{public}}, \Pi_{K,X,d} O_K^{X,d}(\tau))$  (*where  $K \in \{\text{CK}, \text{RK}\}, X \in \{I, R\}, d \in \{s, r\}$  *)

```

Figure 13: The set of oracles $O(\tau)$.

dire as it could seem, as a non-deducible key still yields a RoR secret

key when fed to a random oracle. We have no reason to believe that fixing the proof in [25] would have an impact on the advantage claimed.

E.1 Issues in [25]

We provide more details on the issue we found in [25]. The authors of [25] explicitly state that *e.g.* if the master secret ms , derived during Extended Triple Diffie-Hellman (XDH), is proved secret, then the first root key and chain keys are indistinguishable from random.

Real or random of RK_1 [25, p. 36]

In Case 1 (or Case 2), we argued that all values derived from ms using HKDF were indistinguishable from random.

Therein lies the issue, as at this point, due to the potential compromise of rk_2 , rk_1 is in fact not indistinguishable from random, it is only non-deducible by an attacker.

This argument arises several times in the proof, with a similar omission (as a typo irrelevant to our discussion we note that in the following quote reproduced verbatim, x should have been 1).

Real or random of $ck_{1,1}^{ri}$ [25, p. 36]

After replacing the root key rk_1 , it is straightforward to see that it is impossible for the adversary to differentiate keys derived in this stage—chaining keys $ck_{x,0}^{ri}$ and $ck_{x,1}^{ri}$, messaging key $mk_{x,0}^{ri}$,

and intermediate value tmp from random: these are derived by applying KDFr to rk_1 and then KDFm to that result.

Here, a later $ck_{1,2}^{ri}$ could be leaked, once again breaking the RoR of the previous keys in the chain.

Similarly when looking at any future chain key after several asymmetric ratchets, the situation is similar (now, $1 < x$).

Real or random of RK_{x+1} [25, p. 37]

Treating the KDF as a random oracle, this stage's message key, as well as the next root key rk_{x+1} and the symmetric chaining keys $ck_{x,0}^{ir}$ and $ck_{x,1}^{ir}$, are then indistinguishable from random.

We do not highlight more quotes, but similar arguments are present up to page 45.

E.2 Issues in [17]

Later pen-and-paper proofs of the DR [1, 17, 23, 27] appear to mostly avoid relying on this problematic type of arguments, possibly thanks to the fact that their proofs are modular.

However, we still found another instance of this problem, where a misleading argument omits the fact that a value produced by a random oracle is often not RoR in the context of PCS due to the leakage of later values.

Real or random of σ_{root}^{t*} [17, p. 47]

First note that since we model H as a random oracle, while A has not queried the random oracle on $(sp.\sigma_{root}^{t*-1}, I_t^*)$, the output $(sp.\sigma_{root}^{t*}, k)$ is always uniformly random to A .

This argument suffers from the same issue as before. To see the problem, consider the keys $sp.\sigma_{root}^{t*-1}$ and $sp.\sigma_{root}^{t*}$ discussed in the quote above. Further, consider the next key $sp.\sigma_{root}^{t*+1}$ in the chain of keys. From the expected definition of PCS, $sp.\sigma_{root}^{t*+1}$ could be compromised by the adversary. Here, A can distinguish between $(sp.\sigma_{root}^{t*}, k)$ and a random uniform key k^* by sending $(sp.\sigma_{root}^{t*}, k)$ to the random oracle and checking whether they get $(sp.\sigma_{root}^{t*+1}, k)$, which they know. This attack does not require to send the *previous* key $(sp.\sigma_{root}^{t*-1}, k)$ to the random oracle. This invalidates the argument presented in the quote above.