The Squirrel Prover and its Logic

David Baelde, Univ Rennes, CNRS, IRISA
Stéphanie Delaune, CNRS, Univ Rennes, IRISA
Charlie Jacomme, Université de Lorraine, LORIA, Inria Nancy Grand-Est
Adrien Koutsos, Inria Paris
Joseph Lallemand, CNRS, Univ Rennes, IRISA

The Squirrel system [Baelde et al. 2021] is an interactive prover for the verification of cryptographic protocols. It relies on a dedicated higher-order logic and provides security guarantees akin to those classically obtained by cryptographers, against attackers modeled as arbitrary probabilistic polynomial-time Turing machines. In this paper, we provide a high-level introduction to the logic behind Squirrel, and briefly describe some of the interesting technical challenges encountered during its construction.

1. INTRODUCTION

Security protocols are widely used today to secure transactions that take place over public channels like the Internet. Common uses include the secure transfer of sensitive information such as credit card numbers, or user authentication on a system. Because of their presence in many widely used applications (*e.g.* electronic commerce, government-issued ID), developing methods and tools to verify security protocols has become an important research challenge. Such tools help increase our trust in protocols, and hence on the applications that rely on them.

Formal methods have brought various approaches to prove that cryptographic protocols indeed guarantee the expected security properties. An effective approach in this area of research consists in modeling cryptographic messages as first-order terms, together with an equational theory that represents attacker capabilities. This idea, originally proposed in [Dolev and Yao 1981], has been refined over the years, resulting in a variety of so-called *symbolic* models. These models encompass broad categories of attackers and facilitate the automated verification of protocols. They have led to the development of successful tools such as ProVerif [Blanchet 2001] and Tamarin [Meier et al. 2013]. However, it is important to note that security in a symbolic model does not necessarily imply security in the cryptographers' standard model, called the *computational* model. In that model, attackers are represented by probabilistic polynomial-time Turing machines (PPTMs), and one proves that a protocol is indistinguishable from an idealized, obviously secure version of it. Verification techniques for the computational model, though crucially needed, often exhibit less flexibility or automation compared to ones for symbolic models. As an illustration, secret keys are faithfully modeled in the computational model as long bitstrings that are drawn uniformly at random, whereas they are modeled using abstract names in symbolic models. In symbolic models, two distinct secret keys are represented by different names, which cannot be equal. However, in the computational model, as in reality, it is possible (although unlikely) that the sampled bitstrings are equal. In this column, we present a recent logic-based method for verifying cryptographic protocols in the computational model, and some practical aspects of its implementation in the Squirrel tool [Baelde et al. 2021; Baelde et al. 2023]. This system is built on the Computationally Complete Symbolic Attacker (CCSA) approach of [Bana and Comon-Lundh 2012; Bana and Comon-Lundh 2014], which relies on the symbolic setting of logic, but avoids the limitations of the symbolic models mentioned above. Instead of modeling attacker capabilities by rules stating what the adversary can do, the CCSA method relies on the specification of what the attacker *cannot* do. Starting from the security properties of cryptographic primitives, one derives rules that articulate which pairs of message sequences are indistinguishable. These rules are proved sound w.r.t. the interpretation of terms as PPTMs. Therefore, a proof of a security property using these rules implies security in the computational model under the initial cryptographic assumptions. The CCSA logic was later extended into a meta-logic, which served as basis for the first version of Squirrel [Baelde et al. 2021], before being generalized to a fully-fledged higher-order logic in [Baelde et al. 2023]. The Squirrel tool is a proof assistant developed in a collaborative effort, which has been successfully used on a variety of case studies [Baelde et al. 2022; Comon et al. 2020; Cremers et al. 2022]. Instructions for installing the system are available on its website, together with a user manual, tutorials, and an in-browser interface for playing with the tool without installing it:

https://squirrel-prover.github.io

We first provide, in Section 2, an introduction to the computational model and the CCSA approach, showing in particular how cryptographic assumptions translate into logical rules. We elaborate on this in Section 3 to show how protocols can be modeled in CCSA logic, discussing a subtle issue w.r.t. the intended notion of security. Finally, Section 4 formally defines the higher-order CCSA logic, on which Squirrel is based, and discusses some of its interesting technical features: the distinction between local and global formulas, the subtleties tied to reasoning about probabilistic objects, and the key notion of bi-deduction for reasoning about computational indistinguishability.

Related work. Squirrel is only a recent addition to the list of available systems for verifying cryptographic protocols. We have mentioned above some tools that provide guarantees in symbolic models; we now briefly present the tools providing guarantees in the computational model. Several such systems exist, based on different approaches. The earliest one is CryptoVerif [Blanchet 2008], which mechanizes proofs based on high-level game transformations, following the style of pen-and-paper proofs. The most prominent system today is probably EasyCrypt [Barthe et al. 2011], which is a proof assistant also featuring higher-order logic. It notably embeds a domain-specific *probabilistic relational Hoare logic* [Barthe et al. 2009], which can capture cryptographic game transformations. This design makes it possible to carry out most cryptographic arguments within EasyCrypt. Other systems are currently being developed, with various goals: notably CryptHOL [Basin et al. 2020], which explores an alternative modeling technique in Isabelle/HOL, and F$\star$ [Swamy et al. 2016], which is a general-purpose program verification framework based on refinement types that can be used (via external arguments) to provide computational security guarantees. Those approaches can be compared on several criteria [Barbosa et al. 2021].

Here, we only compare with the closest two tools, and only with respect to three criteria, to highlight differences: modeling, automation, and proof methodology. Regarding modeling, CryptoVerif and Squirrel have a similar level of detail and expressivity, although CryptoVerif supports a larger range of cryptographic assumptions. EasyCrypt is more expressive, with a higher level of detail, at the cost of modeling overhead, and thus better suited for proving properties of primitives. Protocol specifications in CryptoVerif and Squirrel are given in a process algebra, whereas protocols are encoded in EasyCrypt as APIs, which is inconvenient for interactive protocols. Finally, CryptoVerif does not support stateful protocols, while Squirrel and EasyCrypt do.

Overall, the current level of automation of Squirrel sits somewhere between CryptoVerif, which can apply cryptographic arguments automatically, although it often requires hints about which game transformations to use, and EasyCrypt, which does not automatically apply cryptographic arguments.

Finally, the most important difference between Squirrel and the other tools is the associated proof methodology: CryptoVerif relies on game transformations and EasyCrypt performs Hoare-style proofs of programs, while Squirrel reasons over execution traces of protocols.

2. A BASIC INDISTINGUISHABILITY LOGIC

We introduce in this section the CCSA approach, which is designed to reason on cryptographic protocols, *i.e.* concurrent programs relying on cryptographic primitives to achieve some functionality while running in a malicious environment. To simplify our exposition, we delay the treatment of general protocols to Section 3, and restrict our presentation to simple sequential cryptographic games for the time being. These games are used by cryptographers to express security properties of cryptographic primitives against attackers modeled as arbitrary polynomial-time computations. A key concept about games is *computational indistinguishability*: roughly, two games are indistinguishable if any polynomial-time adversary has at-best a negligible probability in distinguishing them. In the CCSA approach, we use first-order logic to establish computational indistinguishability: we first model the messages derived during a game's execution as first-order terms, and then reason about computational indistinguishability in first-order logic, using inference rules derived from cryptographic assumptions.

A key aspect of the logic is that it abstracts cryptographic arguments as much as possible. Notably, its syntax does not mention the length of keys or the value of probabilities. Further, reduction-based arguments are implicit: they justify the soundness of the proof rules, but the code underlying reductions is never explicitly stated.

2.1. The core of the logic

Security in the real world is often conditional, as there is usually a small but non-zero probability that the adversary manages to break the security of a system. Typically, one cannot rule out that a lucky attacker guesses a secret key. To discard such unlikely cases, security is considered up to a *negligible* probability of attack. A function $f : \mathbb{N} \to [0, 1]$ is *negligible* if it is asymptotically lower than the inverse of any polynomial, *i.e.* if $\forall\, k \in \mathbb{N}.\ \exists n_0.\ \forall n \geq n_0.\ f(n) \leq \frac{1}{n^k}$. Conversely, f is *overwhelming* if $1 - f$ is negligible. In what follows, we will thus use a *security parameter* $\eta \in \mathbb{N}$ which can be, *e.g.*, the length of the secret keys, and security must hold with *overwhelming* probability w.r.t. η.

In the cryptographic literature, security properties are commonly specified using so-called *games*, in which an adversary tries to mount an attack: the game might represent an adversary attempting to guess a secret value, to forge a signature, . . . Games are usually written in pseudo-code using an imperative style, as shown in Figure 1. The statement $x \xleftarrow{\$} D$ stores the result of a random sampling following distribution D into variable x: *e.g.* $\mathsf{sk} \xleftarrow{\$} \{0, 1\}^\eta$ uniformly samples a secret key sk of η bits. Games are parameterized by an abstract *interactive attacker*, denoted by $\mathcal{A}$. The statement $o \leftarrow \mathcal{A}(i)$ calls the adversary $\mathcal{A}$ on input i, and stores its answer in variable o. The adversary is stateful, retaining information across its invocations. Standard assignments are denoted in the same way: $x \leftarrow e$ stores the result of evaluating e into variable x.

Describing cryptographic games as probabilistic imperative programs lets cryptographers rely on the reader's intuitive understanding of their semantics, and avoids introducing convoluted execution models. However, this comes at a cost: formally reasoning over such programs can be difficult (one needs to deal with statefulness, loop invariants, probabilistic dependencies, . . .); moreover, encoding complex protocols as imperative programs is possible but not natural, and complicates security proofs.

```
Game $\mathcal{G}$:
$\mathsf{sk} \xleftarrow{\$} \{0,1\}^\eta$;
$m \leftarrow \mathcal{A}()$;
$t \leftarrow \mathsf{enc}(m, \mathsf{sk})$;
$x \leftarrow \mathcal{A}(t)$;
return $x$
```

A secret key sk of length η is randomly sampled. The interactive attacker $\mathcal{A}$ is asked to provide a message m which is then encrypted (using sk) and sent back to $\mathcal{A}$. Afterwards, $\mathcal{A}$ returns a second message x which is the final output of the game.

Fig. 1. An example cryptographic game.

Cryptographic games in CCSA. In the CCSA approach, we do not explicitly represent games. Instead, we shall only represent the messages computed at different points in a game, using first-order terms. Those are pure, unlike the stateful games, and thus easier to reason about. Our terms are built using:

- *honest function symbols* which represent the various primitives used to compute messages (*e.g.* pairing, encryption);
- *attacker function symbols*, noted att_i for $i \in \mathbb{N}$, modeling arbitrary computations performed by adversaries;
- *name symbols*, representing the sources of randomness: essentially, a name is a pointer to a memory cell holding a value sampled at random before the game starts.

Example 2.1. Let us illustrate this approach by modeling the final result of the game of Figure 1 as a single term. The sampling of the secret key is modeled by a name symbol sk. The first call to the attacker $\mathcal{A}()$ is modeled by the term $\mathsf{att}_0()$. In the imperative game modeling style, internal functions (*e.g.* the encryption enc) can be probabilistic, but this is not so in our approach, where we precisely track probabilistic dependencies. Thus, we explicitly specify the randomness used by the encryption function enc, by introducing a name symbol r and using the term $\mathsf{enc}(\mathsf{att}_0(), \mathsf{r}, \mathsf{sk})$ to model t in the game. Then, the second call to the adversary can be modeled using a different adversarial function symbol, as: $\mathsf{att}_1(\mathsf{enc}(\mathsf{att}_0(), \mathsf{r}, \mathsf{sk}))$. Importantly, we have modeled two successive calls to the stateful attacker $\mathcal{A}$ in the game by two pure functions (att_0 and att_1). This is without loss of generality, as any state computed during the first call to $\mathcal{A}$ and used in the second call can be recomputed when modeling the second call as att_1.

Models of the logic. Defining the logic's semantics in order to have a faithful translation from games to terms requires a bit of care. The interpretation of a term is parameterized by the security parameter $\eta \in \mathbb{N}$, as well as two explicit sources of randomness provided by a pair $\rho = (\rho_h, \rho_a)$ of tapes filled with random bits. A model $\mathbb{M}$ of our logic must:

- provide for each honest function symbol f its interpretation as a PPTM $\mathcal{M}_\mathsf{f}$;
- associate to each name n a unique sub-sequence of η bits in ρ_h using a PPTM $\mathcal{M}_\mathsf{n}$ with access to ρ_h, such that distinct names use disjoint parts of ρ_h;
- interpret any adversarial function symbol att as a PPTM $\mathcal{M}_\mathsf{att}$ which can only access the random tape ρ_a (but not ρ_h).

For the sake of simplicity, we forced names to be interpreted as uniform and independent random samplings in $\{0,1\}^\eta$.

Our models are first-order models with a tailored interpretation domain. Thus, terms can be interpreted in $\mathbb{M}$ using the standard semantics of first-order logic. In our specific setting, a term t is interpreted as a function associating, to each value of the security parameter η and random tapes ρ, the interpretation $[\![t]\!]_\mathbb{M}^{\eta,\rho}$. More precisely, the function $\eta, \rho \mapsto [\![t]\!]_\mathbb{M}^{\eta,\rho}$ is computed by a PPTM.

For example, the interpretation $[\![\mathsf{n}]\!]_{\mathbb{M}}^{\eta,\rho}$ of a name symbol n is computed by $\mathcal{M}_{\mathsf{n}}(\eta, \rho_h)$, and an adversarial computation is computed by $[\![\mathsf{att}_i(t)]\!]_{\mathbb{M}}^{\eta,\rho} = \mathcal{M}_{\mathsf{att}_i}(\eta, [\![t]\!]_{\mathbb{M}}^{\eta,\rho}, \rho_a)$. Importantly, this interpretation is compositional and makes all probability dependencies explicit, which will facilitate reasoning over cryptographic messages.

Example 2.2. Coming back to $\mathcal{G}$ from Figure 1, the value in the variable x at the end of $\mathcal{G}$'s execution follows the same probability distribution as

$$[\![\mathsf{att}_1(\mathsf{enc}(\mathsf{att}_0(), \mathsf{r}, \mathsf{sk}))]\!]_{\mathbb{M}}^{\eta,\rho}$$

where the tapes in ρ are sampled at random, and where $\mathbb{M}$ interprets the encryption and attacker functions as does $\mathcal{G}$.

Using Turing machines to interpret terms gives us a high level of expressivity. For instance, conditional branching in games can be internalized in the first-order terms using an (if b then t else e) function symbol whose interpretation is forced to be the natural one in all models.

Example 2.3. Consider the following game:

```
Game $\mathcal{G}_{\mathsf{sk}}$
$\mathsf{sk} \xleftarrow{\$} \{0,1\}^\eta$;
$m \leftarrow \mathcal{A}()$;
if $m = \mathsf{sk}$ then
    return $0$
else
    return $1$
```

This game samples a fresh secret key and queries the attacker to obtain a guess m for the key. The game outputs 0 if the attacker correctly guessed the key, and 1 otherwise. The message returned at the end of the game can be represented by the following term:

$$\phi_{\mathsf{sk}} \stackrel{\text{def}}{=} \text{if } (\mathsf{att}_0() = \mathsf{sk}) \text{ then } 0 \text{ else } 1.$$

2.2. The computational indistinguishability predicate $\sim$

One of the most commonly used notions to define cryptographic properties is *computational indistinguishability*. Roughly, indistinguishability is expressed as a guessing game in which an adversary must figure out which one of two scenarios $\mathcal{G}_I$ or $\mathcal{G}_R$ it is interacting with. More precisely, two games $\mathcal{G}_I$ and $\mathcal{G}_R$ are indistinguishable, which we write $\mathcal{G}_I \approx \mathcal{G}_R$, when any PPTM adversary has at-best a negligible probability of guessing whether it is interacting with $\mathcal{G}_I$ or $\mathcal{G}_R$. Formally, we define the *advantage* of the attacker $\mathcal{A}$ as the probability that it makes the two games behave differently. We then require this advantage to be negligible in the security parameter η:

$$\forall \mathcal{A} \in \mathsf{PPTM}.\ \eta \mapsto |\Pr(\mathcal{G}_I(\mathcal{A}) = 1) - \Pr(\mathcal{G}_R(\mathcal{A}) = 1)| \text{ is negligible.}$$

Typically, the game $\mathcal{G}_R$ will correspond to an attack against the real primitives, while $\mathcal{G}_I$ will represent an attack against an idealized implementation of the primitives for which security is obvious.

Example 2.4. Coming back to Example 2.3, the indistinguishability $\mathcal{G}_{\mathsf{sk}} \approx (\textbf{return } 1)$ states that the probability that the adversary $\mathcal{A}$ guesses the secret key sk is negligible. In that case, a simple probabilistic independence argument can be used to show that the probability that $\mathcal{A}$ computes sk is $\frac{1}{2^\eta}$. Thus, the indistinguishability holds.

Squirrel's logic relies on a predicate $\sim$ to represent computational indistinguishability. Formally, for two terms u, v, the predicate $u \sim v$ holds in a model $\mathbb{M}$ if $\mathcal{G}_u^{\mathbb{M}} \approx \mathcal{G}_v^{\mathbb{M}}$, where $\mathcal{G}_t^{\mathbb{M}}$ is the game where the adversary is provided with $[\![t]\!]_{\mathbb{M}}^{\eta,\rho}$ and must produce a bit b, which the game returns. In other words, $u \sim v$ in $\mathbb{M}$ when:

$$\forall \mathcal{A} \in \mathsf{PPTM}.\ \eta \mapsto |\Pr_\rho(\mathcal{A}([\![u]\!]_{\mathbb{M}}^{\eta,\rho}, \rho_a) = 1) - \Pr_\rho(\mathcal{A}([\![v]\!]_{\mathbb{M}}^{\eta,\rho}, \rho_a) = 1)| \text{ is negligible.}$$

$$\textsc{Refl}\ \dfrac{}{u \sim u} \qquad \textsc{Rewrite}\ \dfrac{(u = v) \sim \mathsf{true} \qquad C[v] \sim w}{C[u] \sim w} \qquad \textsc{Fresh}\ \dfrac{}{(t \neq \mathsf{n}) \sim \mathsf{true}}\ \text{when } t \text{ is a ground term in which } \mathsf{n} \text{ does not occur}$$

Fig. 2. Basic inference rules over $\sim$.

Example 2.5. Reusing $\mathcal{G}_{\mathsf{sk}}$ and ϕ_{sk} from Example 2.3, checking $\phi_{\mathsf{sk}} \sim 1$ amounts to verifying that $\mathcal{G}_{\mathsf{sk}} \approx (\mathbf{return}\ 1)$.

In the example above, we have used the indistinguishability predicate on booleans. In that case, the semantics of $\sim$ can be restated in a simpler way, without a quantification over all distinguishers $\mathcal{A}$: it simply means that the two booleans have the same probability of being 1. The general semantics of $\sim$ becomes useful, though, to reason on it by decomposing terms: for instance, it allows us to derive $f(u) \sim f(v)$ from $u \sim v$ for any u and v of arbitrary types. Indeed, the existence of a PPTM distinguishing between $f(u)$ and $f(v)$ implies the existence of a PPTM distinguishing between u and v: the latter distinguisher is obtained by composing the former with the PPTM computing f.

Logical rules. We now present some rules that can be used to reason over $\sim$. We intuitively describe some of those rules here, starting with the simpler rules that do not rely on any security assumptions over the primitives.

Example 2.6. The following formulas are valid:

(1) $t \sim t$ for any term t; indistinguishability is reflexive.
(2) $\mathsf{n} \sim \mathsf{n}'$ for any names n, n'; two uniform random samplings are indistinguishable.
(3) $(\mathsf{n}, \mathsf{n}) \not\sim (\mathsf{n}, \mathsf{n}')$; the attacker sees, in one case, the same value twice, and in the other, two distinct values (with overwhelming probability). Notice that this implies that $\sim$ does not lift to an arbitrary context: we have seen that $\mathsf{n} \sim \mathsf{n}'$, but this example shows that we do not have for all context C that $C[\mathsf{n}] \sim C[\mathsf{n}']$.
(4) $(\mathsf{if\ true\ then}\ u\ \mathsf{else}\ v = u) \sim \mathsf{true}$ for any terms u, v; more generally, any term occurring in an indistinguishability can be rewritten into an equal term.
(5) $(\mathsf{n} \neq \mathsf{n}') \sim \mathsf{true}$; names, *i.e.* random samplings, collide with negligible probability.

We provide a first set of inference rules in Figure 2. Refl corresponds to the reflexivity of $\sim$, and Rewrite allows replacing two terms that are equal with overwhelming probability in any context. Finally, Fresh exploits the fact that a term syntactically contains all its probabilistic dependencies: if a name n does not occur in a term t, then n is a uniform random sampling independent of t, and thus $t = \mathsf{n}$ can only be true with negligible probability.

Example 2.7. Let us go back to Example 2.5 and the formula $\phi_{\mathsf{sk}} \sim 1$. We assume an additional proof rule $\textsc{Simpl}_{\equiv}$ that allows to replace a term with another one equal to it with probability one, which we note $\equiv$, and where for instance, $v \equiv (\mathsf{if\ false\ then}\ u\ \mathsf{else}\ v)$. We can prove our goal with the following derivation:

$$\textsc{Rewrite}\ \dfrac{\textsc{Simpl}_{\equiv}\ \dfrac{\textsc{Fresh}\ \dfrac{}{(\mathsf{att}_0() \neq \mathsf{sk}) \sim \mathsf{true}}}{((\mathsf{att}_0() = \mathsf{sk}) = \mathsf{false}) \sim \mathsf{true}} \qquad \textsc{Simpl}_{\equiv}\ \dfrac{\textsc{Refl}\ \dfrac{}{1 \sim 1}}{(\mathsf{if\ false\ then}\ 0\ \mathsf{else}\ 1) \sim 1}}{(\mathsf{if\ att}_0() = \mathsf{sk\ then}\ 0\ \mathsf{else}\ 1) \sim 1}$$

Here, we use twice $\textsc{Simpl}_{\equiv}$: once with the previously mentioned equality over a false conditional, and once using the fact that $((\mathsf{att}_0() = \mathsf{sk}) = \mathsf{false}) \equiv (\mathsf{att}_0() \neq \mathsf{sk})$.

2.3. Unforgeability of hash functions: EUF-CMA

To reason over cryptographic games, a final piece of the puzzle is missing: assumptions over cryptographic primitives. For each primitive (a cipher, a signature, a hash, ...), there is a usual set of cryptographic assumptions made over it. Such assumptions are expressed as an indistinguishability between two games, and cryptographic proofs take the form of reduction-based arguments. To prove that the security of the primitive, for instance expressed as $P_R \approx P_I$, implies the security of some game $\mathcal{G}_R \approx \mathcal{G}_I$, we prove the contrapositive: assuming that there is a distinguisher against $\mathcal{G}_R \approx \mathcal{G}_I$, we build a distinguisher against $P_R \approx P_I$. In our logic, we hide those reduction-based arguments behind dedicated rules, one for each cryptographic assumption.

Consider the example of a keyed hash function $\mathsf{h}(x, \mathsf{sk})$ and of the EUF-CMA assumption. Intuitively, a keyed hash function is a function that takes a message and a key, and produces a short unpredictable value. More precisely, if a key sk is secret and randomly sampled, then an attacker has a negligible probability of guessing the hash $\mathsf{h}(m, \mathsf{sk})$ of any m, unless the attacker was directly given this value. Such a hash function is called *unforgeable*, and this is formalized in the $\text{EUF-CMA}_\mathsf{h}$ cryptographic game:

```
EUF-CMA_h
  $\mathsf{sk} \xleftarrow{\$} \{0,1\}^\eta$;
  $\mathcal{L} \leftarrow \emptyset$;
  $t, m \leftarrow \mathcal{A}^O()$;
  return $(t = \mathsf{h}(m, \mathsf{sk}) \wedge m \notin \mathcal{L})$
```

```
Hashing Oracle $O(x)$
  $\mathcal{L} \leftarrow \{x\} \cup \mathcal{L}$;
  return $\mathsf{h}(x, \mathsf{sk})$
```

This game asks the attacker to return a pair of values t, m such that $t = \mathsf{h}(m, \mathsf{sk})$. The attacker is allowed to interact with the hashing oracle O, but it must return a m that was never queried to O. This is enforced by storing in a set $\mathcal{L}$ all oracle inputs.

We can express the security of the hash function by assuming that for all attackers $\mathcal{A}$, $\text{EUF-CMA}_\mathsf{h} \approx (\mathbf{return}\ \mathsf{false})$. Our goal is now to derive an inference rule which is sound under this assumption.

As a warm-up, we restrict ourselves to models where the interpretation of h satisfies the EUF-CMA assumption, and analyze the validity of a few statements:

- $(\mathsf{h}(0, \mathsf{sk}) = \mathsf{att}_0()) \sim \mathsf{false}$ is valid. Otherwise, we would have a model (and thus an attacker) able to output with non-negligible probability the value of $\mathsf{h}(0, \mathsf{sk})$ without having access to any other hash values. This attacker would trivially break $\text{EUF-CMA}_\mathsf{h}$, by directly outputting $0, \mathsf{h}(0, \mathsf{sk})$.
- $(\mathsf{h}(0, \mathsf{sk}) = \mathsf{att}_1(\mathsf{sk})) \sim \mathsf{false}$ is not valid, because the secret key is leaked to the attacker. As such, the computation of $\mathsf{att}_1(\mathsf{sk})$ cannot be seen as a computation made by an adversary $\mathcal{A}^O$ in the $\text{EUF-CMA}_\mathsf{h}$ game, and the assumption does not apply.
- $(\mathsf{h}(0, \mathsf{sk}) = \mathsf{att}_1(\mathsf{h}(1, \mathsf{sk}))) \sim \mathsf{false}$ is valid. Otherwise, we would have a model and thus an attacker breaking $\text{EUF-CMA}_\mathsf{h}$: the attacker would compute $\mathsf{h}(1, \mathsf{sk})$ with a call to oracle $O(1)$, after which $\mathcal{L} = \{1\}$, and return $\mathsf{att}_1(\mathsf{h}(1, \mathsf{sk}))$, which is the hash of $0 \notin \mathcal{L}$.

To generalize this, we need to answer the following question: under which conditions over t and m is $(\mathsf{h}(m, \mathsf{sk}) = t) \sim \mathsf{false}$ a valid formula? We answer this by providing sufficient conditions under which the terms t and m can be produced by some attacker interacting with the $\text{EUF-CMA}_\mathsf{h}$ game in such a way that m is never queried to O.

The first main condition is that all syntactic occurrences of sk in t and m must be as a key to h (*i.e.* all occurrences are of the form $\mathsf{h}(u, \mathsf{sk})$ for some u). Otherwise, the terms cannot be simulated, as the adversary $\mathcal{A}$ against $\text{EUF-CMA}_\mathsf{h}$ does not know the key.

Once we know that sk is only used in key position, it is easy to syntactically track in t and m what will be computed using queries to O: it is precisely the set of subterms

```
Game $\mathcal{G}(\mathscr{R})$
    $\mathsf{sk} \xleftarrow{\$} \{0,1\}^\eta$;
    $\mathcal{L} \leftarrow \emptyset$;
    $b \leftarrow \mathcal{A}^{\mathscr{T},\mathscr{R}}()$;
    return $b$

Tag $\mathscr{T}$
    $\mathsf{n} \xleftarrow{\$} \{0,1\}^\eta$;
    $x \leftarrow \langle \mathsf{n}, \mathsf{h}(\mathsf{n},\mathsf{sk})\rangle$;
    $\mathcal{L} \leftarrow \{x\} \cup \mathcal{L}$;
    return $x$

Reader (real) $\mathscr{R}_r(x)$
    $x_n \leftarrow \mathsf{fst}(x)$;
    $x_h \leftarrow \mathsf{snd}(x)$;
    return $(\mathsf{h}(x_n,\mathsf{sk}) = x_h)$

Reader (ideal) $\mathscr{R}_i(x)$
    return $x \in \mathcal{L}$
```

Fig. 3. Authentication game for the BASIC HASH protocol.

of the form $\mathsf{h}(x,\mathsf{sk})$, *i.e.* $S^{\mathsf{h}}_{\mathsf{sk}}(t,m) = \{\mathsf{h}(u,\mathsf{sk}) \mid \mathsf{h}(u,\mathsf{sk})$ subterm of t or $m\}$. Using this set, we can then define a formula, inside our logic, expressing that all previously seen hash values are distinct from m. Those two conditions are sufficient to obtain a rule enabling us to use EUF-CMA to establish indistinguishabilities in the logic:

$$\textsc{Euf-Cma}\ \frac{\bigwedge_{\mathsf{h}(u,\mathsf{sk})\in S^{\mathsf{h}}_{\mathsf{sk}}(t,m)} ((u=m) \sim \mathsf{false})}{(\mathsf{h}(m,\mathsf{sk}) = t) \sim \mathsf{false}}\quad \mathsf{sk}\ \textit{only occurs as}\ \mathsf{h}(x,\mathsf{sk})\ \textit{in}\ t\ \textit{and}\ m$$

This rule has a logical premise, which becomes a proof obligation when the rule is applied to a proof goal, and a syntactic side condition automatically checked by Squirrel.

3. MODELING INTERACTIVE PROTOCOLS WITH RECURSIVE FUNCTIONS

Now that we have introduced the core CCSA framework, let us turn to how cryptographic protocols are modeled. Compared to the simple games of the previous section, protocols are concurrent systems, involving several communicating agents. An agent may for instance be a key server, a bank, a user's terminal... When analyzing the security of a protocol, it is desirable to make worst-case assumptions. A typical one is that the adversary has full control over the network: it receives all messages output by protocol agents, and is tasked with feeding inputs to them, with messages resulting from arbitrary computations. We also assume that the adversary schedules the actions of the protocol's agents: it decides when to spawn a new session, when to advance in a session, ... This can actually be modeled using cryptographic games, representing the agents as oracles to which the adversary has access. Adversarial computations then induce a sequence of interactions with the protocol's agents, *i.e.* an execution trace of the protocol. Following this style, quantifying over all adversaries implies quantifying over all execution traces. We present this approach in more details in what follows, illustrating it on a simplified version of the BASIC HASH protocol [Brusò et al. 2010].

3.1. An example: BASIC HASH protocol

The BASIC HASH protocol is a simple RFID protocol in which a tag $\mathscr{T}$ tries to authenticate itself to a reader $\mathscr{R}$. This is an access control protocol: *e.g.* the tag may be embedded in an RFID fob, and the reader could guard access to some building. The tag and the reader both share a secret key sk. Whenever the tag wants to authenticate itself to the reader, it samples a random value n, hashes it with sk using an unforgeable keyed hash function as presented in Section 2.3, and sends the pair $\langle \mathsf{n}, \mathsf{h}(\mathsf{n},\mathsf{sk})\rangle$ of the name and its hash digest to the reader. The reader can check that a message x it received from the network was generated by the tag by extracting the first component $\mathsf{fst}(x)$ of x, re-computing its hash using the secret key sk and checking that this yields the second component $\mathsf{snd}(x)$ of its input.

Authentication game. We can express the fact that this protocol provides some form of authentication using the cryptographic game presented in Figure 3. The definitions of $\mathscr{T}$ and $\mathscr{R}_r$ correspond to the tag and reader agents described above, with a single addition that will be useful to express the security property: before sending its output

$x = \langle \mathsf{n}, \mathsf{h}(\mathsf{n}, \mathsf{sk})\rangle$ to the reader, the tag logs x into a set $\mathcal{L}$. Further, $\mathscr{R}_i$ corresponds to an idealized reader, which checks that its input x originates from $\mathscr{T}$ by directly verifying that x appears in the log $\mathcal{L}$. Intuitively, the idealized reader looks, across space and time, into the past internal memories of the tag to check whether it indeed generated x. Obviously, $\mathscr{R}_i$ is only useful as a trick to model the expected authentication property, and cannot be physically implemented.

The cryptographic game $\mathcal{G}(\mathscr{R})$, parameterized by the reader $\mathscr{R} \in \{\mathscr{R}_r, \mathscr{R}_i\}$, samples the secret key sk, initializes the log $\mathcal{L}$, and runs the adversary $\mathcal{A}^{\mathscr{T},\mathscr{R}}$ by letting it interact with the tag and reader oracles. The adversary must try to guess a bit b indicating whether it is interacting with the real or ideal reader $\mathscr{R}$: its guess b is the final result of the game. The protocol is secure if $\mathcal{A}$ has a negligible advantage in guessing the correct bit b, *i.e.* if the computational indistinguishability $\mathcal{G}(\mathscr{R}_r) \approx \mathcal{G}(\mathscr{R}_i)$ holds. Indeed, if an attacker can make $\mathscr{R}_r$ accept while breaking authentication (*i.e.* no corresponding tag produced the message), trying to do the same execution with $\mathscr{R}_i$ would fail by construction, making it trivial to distinguish the two worlds.

3.2. Modeling protocol executions

During its execution, $\mathcal{A}$ can call the tag and reader oracles any number of times and according to any interleaving. We are going to model the interaction of $\mathcal{A}$ with the protocol agents $\mathscr{T}, \mathscr{R}$ along an execution trace tr representing a fixed but arbitrary interleaving. Thus, an *execution trace* tr is a finite sequence $\tau_1, \dots, \tau_n$ of *timestamps*, where the j-th timestamp τ_j in the trace represents the agent the adversary interacted with at this step. To distinguish multiple interactions with the same agents, actions are indexed by a unique replication index i in some set of indices $\mathcal{I}$. For BASIC HASH, we use $\tau = \mathsf{T}(i)$ and $\tau = \mathsf{R}(i)$ for an interaction with respectively the tag and the reader. For convenience, we also force τ_1 to a special value init.

$$\mathcal{T} ::= \tau_1, \dots, \tau_n \qquad \tau_i ::= \mathsf{init} \mid \mathsf{T}(i) \mid \mathsf{R}(i) \qquad (n \in \mathbb{N}, i \in \mathcal{I})$$

Modeling execution traces. We assume a typed logic, to enable reasoning over different kinds of objects. Models of the logic provide an interpretation domain for each type, where for instance $\mathsf{message}$ is interpreted as the set of bitstrings and bool as $\{0, 1\}$. Intuitively, the type of all terms used in Section 2 is $\mathsf{message}$ or bool, with *e.g.* the equality function symbol $=$ typed as $\mathsf{message} \to \mathsf{message} \to \mathsf{bool}$. The types index and $\mathsf{timestamp}$ respectively represent the sets of indices $\mathcal{I}$ and timestamps $\mathcal{T}$, and we use function symbols T and R, both of type $\mathsf{index} \to \mathsf{timestamp}$, to build timestamps from indices, and $\mathsf{init} : \mathsf{timestamp}$ for the initial timestamp. The execution trace tr is implicitly encoded using two function symbols:

$$\mathsf{happens} : \mathsf{timestamp} \to \mathsf{bool} \qquad \cdot < \cdot : \mathsf{timestamp} \to \mathsf{timestamp} \to \mathsf{bool}$$

where $\mathsf{happens}(\tau)$ states that τ has been scheduled in the execution trace, and $\tau < \tau'$ that τ occurred before τ' in the trace. Each model $\mathbb{M}$ of the logic fixes the execution trace by interpreting the types and function symbols above. We require, through an ad hoc restriction on models of the logic, that the interpretation of index in any model is finite and independent of the security parameter η. Further, we assume that any scheduled timestamp uniquely corresponds to either init, some $\mathsf{T}(i)$ or some $\mathsf{R}(i)$.

Modeling the game's execution. In the logic, the interaction of the adversary $\mathcal{A}$ with the oracles $\mathscr{T}$ and $\mathscr{R}$ along an execution trace is modeled using several mutually recursive functions called *macros*. A macro is identified by its name m and is evaluated at a particular timestamp τ: informally, $\mathsf{m}@\tau$ refers to the value of m at point τ in the execution. We will use the following macros: $\mathsf{input}@\tau$ and $\mathsf{output}@\tau$ denote the messages input and output by the oracle scheduled at time τ in the execution trace;

and $\mathsf{frame}@\tau$ is roughly the sequence of all messages output by any agent during the execution up to timestamp τ. The frame is crucial to express security properties, as it represents the full list of messages seen by the attacker. We assume a function symbol $\mathsf{pred} : \mathsf{timestamp} \to \mathsf{timestamp}$ that maps any scheduled timestamp different from init to its predecessor in the execution trace, and all other timestamps to a fixed constant undef, distinct from all scheduled timestamps. The macros can be defined as follows for the game $\mathcal{G}(\mathscr{R}_r)$ using the real reader:

$$\begin{aligned}
\mathsf{input_r}@\tau &= \mathsf{match}\ \tau\ \mathsf{with} & \mathsf{frame_r}@\tau &= \mathsf{match}\ \tau\ \mathsf{with}\\
&\quad \mid \mathsf{init} \to \mathsf{empty} & &\quad \mid \mathsf{init} \to \mathsf{empty}\\
&\quad \mid \_ \to \mathsf{att}_\tau(\mathsf{frame_r}@\mathsf{pred}(\tau)) & &\quad \mid \_ \to \langle \mathsf{frame_r}@\mathsf{pred}(\tau), \mathsf{output_r}@\tau \rangle
\end{aligned}$$

$$\begin{aligned}
\mathsf{output_r}@\tau &= \mathsf{match}\ \tau\ \mathsf{with}\\
&\quad \mid \mathsf{init} \to \mathsf{empty}\\
&\quad \mid \mathsf{T}(i) \to \langle \mathsf{n}(i), \mathsf{h}(\mathsf{n}(i), \mathsf{sk}) \rangle\\
&\quad \mid \mathsf{R}(i) \to \mathsf{h}(\mathsf{fst}(\mathsf{input_r}@\tau), \mathsf{sk}) = \mathsf{snd}(\mathsf{input_r}@\tau)
\end{aligned}$$

where empty is a symbol of type $\mathsf{message}$ representing the empty bitstring, and where we generalized names to be of type $\mathsf{index} \to \mathsf{message}$, so that each $\mathsf{n}(i)$ is an independent fresh name. The r subscript indicates that these macros are for the game using the real reader. The macros definitions for the game with the ideal reader $\mathcal{G}(\mathscr{R}_i)$, subscripted by i, are identical except for the output macro: we only need to introduce a macro $\mathcal{L}@\tau$ for the value of the log $\mathcal{L}$ *after* the execution of the oracle at τ, and to modify the $\mathsf{R}(i)$ case of output:

$$\begin{aligned}
\mathsf{output_i}@\tau &= \mathsf{match}\ \tau\ \mathsf{with} & \mathcal{L}@\tau &= \mathsf{match}\ \tau\ \mathsf{with}\\
&\quad \mid \mathsf{R}(i) \to \mathsf{input_i}@\tau \in \mathcal{L}@\tau & &\quad \mid \mathsf{init} \to \emptyset\\
&\quad \mid \ \ldots \to \ldots\ \text{(as before)} & &\quad \mid \mathsf{T}(i) \to \{\mathsf{output_i}@\tau\} \cup \mathcal{L}@\mathsf{pred}(\tau)\\
& & &\quad \mid \mathsf{R}(i) \to \mathcal{L}@\mathsf{pred}(\tau)
\end{aligned}$$

Then, the formula $\mathsf{frame_r}@\tau \sim \mathsf{frame_i}@\tau$ expresses the fact that, for any model $\mathbb{M}$ defining an execution trace tr and an adversary $\mathcal{A}$, the adversary has a negligible probability of distinguishing the real and ideal versions of the protocol, *i.e.* $\mathcal{G}(\mathscr{R}_r) \approx \mathcal{G}(\mathscr{R}_i)$ holds *along this particular execution trace* (see Section 3.3 below for a discussion of the security guarantees provided by this formula).

Cryptographic reasoning with recursive functions. The EUF-CMA reasoning rule presented in Section 2.3 does not directly apply to terms containing recursive definitions: we need to be able to reason over the occurrences of the hash function inside terms, and thus possibly inside an arbitrary recursive unfolding. A first way to fix the EUF-CMA rule is to forbid any occurrence of a hash in all the recursive functions occurring inside a term, but it is of course too limiting: in BASIC HASH, we have for instance a set of hash occurrences of the form $\mathsf{h}(\mathsf{n}(i), \mathsf{sk})$ in $\mathsf{output_r}$. We need to be able to reason over such a set, for instance using some universal quantification over indices or timestamps. We formally introduce such constructions in Section 4.2, and extend the cryptographic rules in Section 4.4.

Implementation in Squirrel. The encoding of protocols using macros as described above can be generalized to support a large class of security protocols. Given a protocol $\mathcal{P}$, we can use the same definitions for frame and input, and we only need to adapt the definition of output according to the protocol specification, and, if necessary, introduce additional macros for the protocol's mutable variables (such as the macro $\mathcal{L}$ for the log $\mathcal{L}$). This method is expressive enough to encode complex protocol behaviors, and allows all relevant information to be expressed as messages that our logic can handle.

However, it is arguably not very intuitive to manipulate for users. To help with that, Squirrel allows users to describe protocols as processes in a process algebra (a variant of the applied π-calculus), and automatically translates them as macros (the actual definition of Squirrel macros is slightly more involved than the one presented here, we refer readers to [Baelde et al. 2022] for details).

3.3. The issue of asymptotic security

With the modeling style described here for protocols, we always reason on an arbitrary but fixed trace. When proving a property, we show that it holds (up to negligible probability) in all models of the logic, and the model includes the interpretation of the order on timestamps, *i.e.* the interleaving of actions, as well as the interpretation of all attacker function symbols. In particular, this means that in any given model, the trace and the interpretation of adversarial functions is fixed, and only then do the security parameter η and random tape ρ vary. In other words, when proving a property ϕ, what we establish can informally be described by the formula:

$$\forall \text{ trace } \mathsf{tr}.\ \forall \text{ adversary } \mathcal{A}.\ \mathrm{Pr}_\rho(\phi \text{ holds in } \mathsf{tr} \text{ against } \mathcal{A}) \text{ is overwhelming in } \eta.$$

While such a statement gives good security guarantees, it is weaker than what is commonly expected. In cryptographic games, the adversary is indeed given access to the protocol through oracles it may call at will any (polynomial) number of times. That is, the adversary can *choose* the trace, rather than having it imposed on him. In particular, the length of the trace may be chosen by the adversary and thus depend on η, while it is fixed in our model. That would correspond to the stronger (informal) formula:

$$\forall \text{ adversary } \mathcal{A}.\ \mathrm{Pr}_\rho(\phi \text{ holds in } \mathsf{tr} \text{ chosen by } \mathcal{A}) \text{ is overwhelming in } \eta.$$

It is possible to overcome this limitation by changing the way we model protocols, to let the adversary adaptively choose the trace during the protocol execution. However, such alternatives have not yet been explored in practice.

4. THE GENERAL LOGIC & PROOF SYSTEM

We have considered so far a typed language with first-order terms and a single indistinguishability predicate, with inference rules that can be expressed as axioms in first-order logic. Except for the addition of recursively defined functions, all this fits within the original CCSA logic of [Bana and Comon-Lundh 2014], which is plain first-order logic. Thus, proving statements in that logic can be done using any proof system for classical first-order logic; only the CCSA axiom schemes would reflect the intended probabilistic semantics of terms and the cryptographic assumptions on primitives.

We have seen, however, that syntactic side conditions, which are needed for cryptographic rules, cannot be easily lifted to terms containing recursive definitions. In addition, the logic appears ill-suited for some security properties. Recall the BASIC HASH example, whose security we defined by using a real and an ideal system. Yet, the intuitive security property should only be about a high-level fact verified by all executions of the real system: whenever a reader accepts, the value it received was produced by an honest tag.

To improve this aspect of the logic, a core idea is to introduce a more specific logic, where special status is given to terms of type bool. We have seen axioms involving indistinguishability, of the form $t \sim \mathsf{true}$ or $t \sim \mathsf{false}$. For such formulas, using computational indistinguishability for all PPTM distinguishers is overkill, and we can take a much simpler approach: $t \sim \mathsf{true}$ simply means that t is true with overwhelming probability. Further, it is convenient to view t itself as a specific kind of formula: to this effect, we can use function symbols representing propositional connectives, as we did for the if-then-else construction; less obviously, the same can be done for quantifiers. In that context, t will be called a *local* formula, while the real first-order formulas are called *global*. It turns out that CCSA proofs tend to intensively reason on local formulas. This calls for a proof system that conveniently handles these like proper formulas. Squirrel comes with one such proof system, which we present in this section.

4.1. Higher-order terms with a relaxed semantics

Before properly introducing local and global formulas and their proof systems, we must come back to the semantics of terms. Indeed, since [Baelde et al. 2023], the logic behind Squirrel features higher-order terms. There have been several motivations for this significant generalization of the original CCSA logic. First, Squirrel developments predominantly consist of proofs of local formulas, and higher-order features at this level help structure these proofs in a modular way, and re-use them. Second, the constraints of the original CCSA logic, namely that all terms are interpreted as PPTMs, make sense for modeling cryptographic protocols and adversaries but are sometimes too restrictive when writing proofs. For instance, it can be useful to talk about the discrete logarithm over finite groups, even though this operation is (hopefully) not PTIME. It can also be useful to quantify over infinite types in local formulas, *e.g.* to specify properties of primitives over messages: anticipating on what follows, we will be able to write local formulas such as $\forall x, y : \mathsf{message}.\ \mathsf{fst}(\langle x, y\rangle) = x$.

The terms of our logic are thus simply-typed λ-calculus terms, additionally featuring recursive definitions. In the tool, some limited forms of polymorphism are even available, but we leave them unaccounted for in the theory. Each type τ is interpreted in a model $\mathbb{M}$ as a family of sets $[\![\tau]\!]^\eta_\mathbb{M}$ indexed by the security parameter η, with $[\![\tau_1 \to \tau_2]\!]^\eta_\mathbb{M} = [\![\tau_1]\!]^\eta_\mathbb{M} \to [\![\tau_2]\!]^\eta_\mathbb{M}$. We require that $[\![\mathsf{bool}]\!]^\eta_\mathbb{M} = \{0, 1\}$ and $[\![\mathsf{message}]\!]^\eta_\mathbb{M} = \{0, 1\}^*$ for all $\mathbb{M}$ and η. The potential dependency in η is however crucial *e.g.* to faithfully model the finite groups used in Diffie-Hellman key exchanges.

Terms are no longer interpreted as PPTMs, but more generally as discrete random variables, *i.e.* functions from random tapes to the desired domain. More precisely, a model fixes a set of *finite tapes* $\mathbb{T}_{\mathbb{M},\eta}$ for each η, and we let $\mathbb{RV}_\mathbb{M}(\tau)$ be the set of all η-indexed sequences of functions from $\mathbb{T}_{\mathbb{M},\eta}$ to $[\![\tau]\!]^\eta_\mathbb{M}$. Then, given $\mathbb{M}$, η and ρ, any term t of type τ is interpreted as $(\rho \in \mathbb{T}_{\mathbb{M},\eta} \mapsto [\![t]\!]^{\eta,\rho}_\mathbb{M})_{\eta\in\mathbb{N}} \in \mathbb{RV}_\mathbb{M}(\tau)$ as follows:

$$\begin{aligned} [\![x]\!]^{\eta,\rho}_\mathbb{M} &= \mathbb{M}(x)(\eta, \rho) \\ [\![t\ t']\!]^{\eta,\rho}_\mathbb{M} &= [\![t]\!]^{\eta,\rho}_\mathbb{M}([\![t']\!]^{\eta,\rho}_\mathbb{M}) \\ [\![\lambda(x : \tau').t]\!]^{\eta,\rho}_\mathbb{M} &= (a \in [\![\tau']\!]^\eta_\mathbb{M} \mapsto [\![t]\!]^{\eta,\rho}_{\mathbb{M}[x/a]}) \end{aligned}$$

where $\mathbb{M}(x)$ is the interpretation of x in $\mathbb{M}$, and $\mathbb{M}[x/a]$ is the model modified to interpret x as a random variable X such that $X(\eta, \rho) = a$. Note that the interpretation of a term only depends on the interpretation of its subterms for the same values of η and ρ, and thus the value of $\mathbb{M}[x/a](x)$ is only relevant on η, ρ. We refer the reader to [Baelde et al. 2023] for the full details, including the interpretation of recursive definitions when they satisfy a suitable well-foundedness condition, as well as how to check that some terms do correspond to PPTM computations in order to apply cryptographic assumptions.

While it could seem more natural to have infinite tapes, rather than finite but arbitrarily long ones, this has technical motivations. It ensures that all functions computed on tapes actually correspond to random variables, *i.e.* that they are measurable. This restriction of our logic (and of the original CCSA logic) is not limiting when it comes to modeling cryptographic protocols and attackers, which all run in PTIME. It is however crucial that the length of tapes can grow with η. The restriction to finite tapes is standard in formal cryptographic reasoning: it is present, *e.g.*, in approaches based on Hoare logics [Barthe et al. 2011; Petcher and Morrisett 2015; Basin et al. 2020].

4.2. Local and global formulas

Squirrel's logic is structured around two kinds of formulas: we call *global* formulas the formulas of first-order logic built around the $\sim$ predicate, and *local* formulas the terms of type bool. To clearly distinguish the two, we add tildes to logical connectives and quantifiers at the global level. We also note formulas with different letters: ϕ and ψ for local formulas and F and G for global ones.

$$\begin{aligned} F &:= F \mathrel{\tilde{\wedge}} F' \mid F \mathrel{\tilde{\vee}} F' \mid F \mathrel{\tilde{\Rightarrow}} F' \mid \tilde{\neg}\, F \mid \tilde{\forall} x.F \mid \tilde{\exists} x.F \mid \vec{u} \sim \vec{v} \mid [\phi] \mid \ldots \\ \phi &:= \phi \wedge \phi' \mid \phi \vee \phi' \mid \phi \Rightarrow \phi' \mid \neg\phi \mid \forall x.\phi \mid \exists x.\phi \mid u = v \mid \ldots \end{aligned}$$

Atomic global formulas include indistinguishabilities, but also atoms of the form $[\phi]$, which can be understood as $\phi \sim \mathsf{true}$. The syntax above is open-ended, as more predicates will be used: in addition to custom predicates introduced by the user, *e.g.* to model verifications on messages, we will make use of several extra predicates later in this section. Note, however, that we will not use equality at the global level.

Indistinguishability atoms $\vec{u} \sim \vec{v}$ can only be formed when u_i and v_i have the same type for all i. Moreover, this type must be of order at most 1, *i.e.* of the form $\tau_1 \to \ldots \to \tau_n$ where all τ_k are base atomic types. Further, we assume that elements of base types can be encoded as bitstrings, which allows interpreting these atoms as in Section 2.2, with a generalization: we now consider a distinguisher that can access the (semantics of) terms as oracles, which can be called on arbitrary inputs to obtain new data.

As said before, local formulas are just boolean terms. As such, the syntax above should be understood as syntactic sugar: for instance, $\phi \wedge \psi$ stands for $(\wedge)\ \phi\ \psi$ where $(\wedge)$ is a constant function of type $\mathsf{bool} \to \mathsf{bool} \to \mathsf{bool}$. We also view local quantifiers in this way: $\forall (x : \tau).\ \phi$ is a notation for $\forall_\tau\ (\lambda(x : \tau).\ \phi)$ where $\forall_\tau$ is a constant of type $(\tau \to \mathsf{bool}) \to \mathsf{bool}$. We require that all models interpret these logical constants as expected. In particular, $[\![\forall x.\ \phi]\!]^{\eta,\rho}_{\mathbb{M}} = 1$ iff $[\![\phi]\!]^{\eta,\rho}_{\mathbb{M}[x/a]} = 1$ for all $a \in [\![\tau]\!]^{\eta}_{\mathbb{M}}$.

Example 4.1. Consider once again the BASIC HASH protocol of Figure 3. Instead of expressing its security by using an equivalence, we can also state it with a local formula over the previously defined macros $\mathsf{input_r}$ and $\mathsf{output_r}$:

$$\begin{aligned} &\forall i : \mathsf{index}.\ \mathsf{happens}(\mathsf{R}(i)) \wedge \mathsf{output_r}@\mathsf{R}(i) = \mathsf{true} \\ &\qquad\qquad \Rightarrow \exists j : \mathsf{index}.\ \mathsf{T}(j) < \mathsf{R}(i) \wedge\ \mathsf{input_r}@\mathsf{R}(i) = \mathsf{output_r}@\mathsf{T}(j) \end{aligned}$$

Asking that this formula holds with overwhelming probability properly expresses authentication for any trace: whenever some tag $\mathsf{R}(i)$ accepts, its input value must have been produced in the past by an honest reader $\mathsf{T}(j)$.

As usual in first-order logic, a global formula can be satisfied (or not) in a model: we write $\mathbb{M} \models F$ when this is the case. A global formula is valid when it is satisfied in all models. We write $F \models G$ if, for all $\mathbb{M}$, $\mathbb{M} \models F$ implies $\mathbb{M} \models G$. A local formula, however, is valid if it is overwhelmingly true in all models: ϕ is valid iff $[\phi]$ is valid, *i.e.* $\eta \mapsto \mathrm{Pr}_{\rho \in \mathbb{T}_{\mathbb{M},\eta}}([\![\phi]\!]^{\eta,\rho}_{\mathbb{M}} = 1)$ is overwhelming.

Example 4.2. We have $[\phi \Rightarrow \psi] \models [\phi] \mathrel{\tilde{\Rightarrow}} [\psi]$: if both $\phi \Rightarrow \psi$ and ϕ are overwhelmingly true, then so must be ψ. However, $[\phi] \mathrel{\tilde{\Rightarrow}} [\psi] \not\models [\phi \Rightarrow \psi]$: indeed, $[\phi] \mathrel{\tilde{\Rightarrow}} [\psi]$ might be satisfied in a model where ϕ is not overwhelmingly true, but true for half of the tapes; hence ψ could be always false, and $[\phi \Rightarrow \psi]$ does not hold.

Example 4.3. We have $[\phi] \mathrel{\tilde{\vee}} [\psi] \models [\phi \vee \psi]$ but not the converse: there might be models where only ϕ holds for half of the random tapes and only ψ holds for the other half, hence $\phi \vee \psi$ is overwhelmingly true (even exactly true) while neither ϕ nor ψ is.

In each model, global formulas are either fixed to true or false, but their quantifiers range over random variables, *i.e.* elements of $\mathbb{RV}_{\mathbb{M}}(\tau)$ for some τ. In contrast, local formulas are probabilistic and interpreted in $\mathbb{RV}_{\mathbb{M}}(\mathsf{bool})$, but their quantifiers range over individuals, *i.e.* elements $a \in [\![\tau]\!]_{\mathbb{M}}^{\eta;\rho}$. Despite this essential difference, we can relate quantifiers at the two levels, as shown in [Baelde et al. 2023, Proposition 2].

PROPOSITION 4.4. *For all $\mathbb{M}$, we have $\mathbb{M} \models [\forall(x : \tau).\phi]$ iff. $\mathbb{M} \models \tilde{\forall}(x : \tau).\ [\phi]$, and similarly for the existential quantifiers.*

4.3. Proof system for global and local reasoning

Squirrel's proof system is given in a natural deduction style (though the tool's tactics are sometimes closer to sequent calculus rules) and it deals with two kinds of sequents. Global sequents are of the form $\mathcal{E}; \Theta \vdash F$ where F is a global formula, Θ is a set of global formulas, and $\mathcal{E}$ is a list of typed variable declarations containing at least all free variables of Θ, F. Such a sequent can be read as the global formula $\tilde{\forall}\mathcal{E}.(\tilde{\wedge}\Theta \tilde{\Rightarrow} F)$, and it is valid when the associated formula is valid. Local sequents are of the form $\mathcal{E}; \Theta; \Gamma \vdash \phi$ where ϕ is a local formula, Γ is a set of local formulas, and the other components are as before, with $\mathcal{E}$ declaring all free variables of Θ, Γ and ϕ. The formula associated to a local sequent is $\tilde{\forall}\mathcal{E}.(\tilde{\wedge}\Theta \tilde{\Rightarrow} [\wedge\Gamma \Rightarrow \phi])$.

It is important to note, in the meaning of local sequents, that the second implication happens at the local level. As a result, the *global* hypotheses of Θ and the *local* hypotheses of Γ take a different meaning, illustrated in the following selected rules:

$$\frac{}{\mathcal{E}; \Theta, F \vdash F} \qquad \frac{}{\mathcal{E}; \Theta; \Gamma, \phi \vdash \phi} \qquad \frac{}{\mathcal{E}; \Theta, [\phi]; \Gamma \vdash \phi} \qquad \frac{\mathcal{E}; \Theta; \emptyset \vdash \phi}{\mathcal{E}; \Theta \vdash [\phi]}$$

The first rule is the usual axiom rule for the global logic. The second and third rules are local and global versions of the axiom rule for the local logic. The last rule allows proving a global sequent whose conclusion is $[\phi]$ from the corresponding local sequent.

The local equality predicate gives rise to the ability to rewrite at the local level, which is in fact a generalization of the REWRITE rule from Figure 2:

$$\frac{\mathcal{E}; \Theta; \Gamma \vdash u = v \quad \mathcal{E}; \Theta; \Gamma \vdash \phi\{x \mapsto u\}}{\mathcal{E}; \Theta; \Gamma \vdash \phi\{x \mapsto v\}}$$

In the tool, a powerful `rewrite` tactic is provided, which builds on the previous rule as well as more basic rules, to perform various kinds of rewriting. The tactic can be invoked on a goal $\mathcal{E}; \Theta; \Gamma \vdash \phi$ to selectively replace occurrences of u into v inside Θ, Γ and ϕ when $u = v$ is part of the hypotheses (or can be simply derived from them). If $u = v$ is a local hypothesis, however, rewriting will only be possible in Γ and ϕ; rewriting in Θ requires a global hypothesis $[u = v]$, and is only possible at specific occurrences in Θ.

The previous result relating global and local quantifiers justifies that our sequents only feature a top-level environment $\mathcal{E}$ corresponding to a global quantification. The rules for universal quantifiers in our proof system are as follows, assuming that x does not occur in $\mathcal{E}$, and that t is a well-typed term of type τ in environment $\mathcal{E}$:

$$\frac{\mathcal{E}, x : \tau; \Theta \vdash F}{\mathcal{E}; \Theta \vdash \tilde{\forall}(x : \tau).F} \qquad \frac{\mathcal{E}; \Theta \vdash \tilde{\forall}(x : \tau).F}{\mathcal{E}; \Theta \vdash F\{x \mapsto t\}} \qquad \frac{\mathcal{E}, x : \tau; \Theta; \Gamma \vdash \phi}{\mathcal{E}; \Theta; \Gamma \vdash \forall(x : \tau).\phi} \qquad \frac{\mathcal{E}; \Theta; \Gamma \vdash \forall(x : \tau).\phi}{\mathcal{E}; \Theta; \Gamma \vdash \phi\{x \mapsto t\}}$$

Let us mention one last articulation between the global and local logics:

$$\frac{\mathcal{E}; \Theta; \Gamma_1 \vdash \phi_1 \quad \mathcal{E}; \Theta \vdash (\wedge\Gamma_0 \Rightarrow \phi_0) \sim (\wedge\Gamma_1 \Rightarrow \phi_1)}{\mathcal{E}; \Theta; \Gamma_0 \vdash \phi_0}$$

In the tool, this rule is made available through a convenient `rewrite equiv` tactic, typically used to establish local formulas when they essentially derive from indistinguishability assumptions.

Example 4.5. A common assumption over hash functions, stronger than EUF-CMA, states that they can be seen as pseudo-random functions (PRF): for a fresh secret key sk, an attacker cannot distinguish between $x \mapsto \mathsf{h}(x, \mathsf{sk})$ and a fully random function. It notably yields in our logic that $\mathsf{h}(0, \mathsf{sk}), \mathsf{h}(1, \mathsf{sk}) \sim \mathsf{n}, \mathsf{m}$. Applying on both sides of the equivalence the function $\lambda x, y : \mathsf{message}.\ x \neq y$, we can obtain the validity of the indistinguishability formula $\mathsf{h}(0, \mathsf{sk}) \neq \mathsf{h}(1, \mathsf{sk}) \sim \mathsf{n} \neq \mathsf{m}$. With this global formula and the `rewrite equiv` tactic, we can reduce the proof of the local sequent $\mathcal{E}; \Theta; \cdot \vdash \mathsf{h}(0, \mathsf{sk}) \neq \mathsf{h}(1, \mathsf{sk})$ to the proof of $\mathcal{E}; \Theta; \cdot \vdash \mathsf{n} \neq \mathsf{m}$.

A more complete description of our proof system is given in [Baelde et al. 2023]. However, it is important to point out that the system presented in that paper (or any earlier one) is only a collection of correct rules: it does not come with any completeness or cut-elimination result. Ongoing investigations are starting to provide such proof-theoretical results, though in a propositional fragment that allows to view our two-level logic in the richer framework of modal logic.

4.4. Cryptographic reasoning with recursive functions

It is natural to lift cryptographic reasoning to either local or global reasoning depending on the nature of the cryptographic assumption. In the case of EUF-CMA from Section 2.3, it could become a local rule of the form:

$$\textsc{Euf-Cma}\ \frac{\mathcal{E}; \Theta; \Gamma \vdash \bigwedge_{\mathsf{h}(u,\mathsf{sk}) \in S^{\mathsf{h}}_{\mathsf{sk}}(t,m)} (u \neq m)}{\mathcal{E}; \Theta; \Gamma \vdash \mathsf{h}(m, \mathsf{sk}) \neq t}\ \mathsf{sk}\ \textit{only occurs as}\ \mathsf{h}(x, \mathsf{sk})\ \textit{in}\ t\ \textit{and}\ m$$

Yet, now that terms contain recursive functions, the meaning of forbidding occurrences of sk is unclear: forbidden computations (*e.g.* a leak of a secret key) might occur deep inside the recursive functions. Further, sk may be an indexed key $\mathsf{k}(i)$, and occurrences with distinct indices $\mathsf{k}(j)$ for $j \neq i$ should be ignored. Checking for such indirect occurrences requires a deeper analysis of the terms involved.

Example 4.6. Let $\mathsf{k}(i)$ be a name symbol indexed by an integer i and representing a key, and consider the following recursive function:

$$\mathsf{keys} = \lambda(i : \mathsf{index}).\ \mathsf{if}\ (i = 0)\ \mathsf{then\ empty\ else}\ \langle \mathsf{k}(i), \mathsf{keys}(i - 1) \rangle$$

where we assume to have functions symbols for the standard operations over integers. Basically, $\mathsf{keys}(i)$ computes the list $[\mathsf{k}(i), \mathsf{k}(i-1), \ldots, \mathsf{k}(1)]$ (encoded as nested pairs). Consider some index terms i_0, j_0 and assume that $j_0 < i_0$. To prove $\mathsf{h}(\mathsf{att}(\mathsf{keys}(j_0)), \mathsf{k}(i_0)) \neq t$ with EUF-CMA, we need to check, among other things, that $\mathsf{k}(i_0)$ does not occur in $\mathsf{keys}(j_0)$. The term $\mathsf{keys}(j_0)$ cannot be evaluated without providing a concrete value for j_0. Moreover, if we attempted to directly check the condition on $\mathsf{keys}(j_0)$'s body, we would find that it contains $\mathsf{k}(i)$ (for a bound i), which can *a priori* be equal to $\mathsf{k}(i_0)$. To conclude here, we need to exploit the hypothesis that $j_0 < i_0$ and details on how the recursive function keys is defined. Thus, we need a more involved check exploiting additional information that cannot be directly obtained from a superficial scrutiny of the terms.

Generalized subterms. We therefore introduce a generalized notion of subterm, to compute an over-approximation of all the subterms that may occur during the evaluation of a recursive function, for any model and random tapes. In addition, to allow exploiting semantic information (*e.g.* the fact that for any $\mathsf{k}(i)$ in $\mathsf{keys}(j)$, we have $i \neq 0$), we gather together with each subterm a local formula expressing the condition under which that subterm may be computed. A generalized subterm, or *occurrence*, σ in a term t is a tuple $(\mathcal{V}, \phi, s)$: s is, roughly, a subterm of t (expanding recursive definitions), $\mathcal{V}$ is a set of typed variables bound above s in t, and ϕ is the condition under which s is evaluated in t, gathered from all if conditions. We use this notion, denoted $\mathsf{ST}(t)$, in cryptographic rules to replace the previous conditions on syntactic subterms. For instance, a condition expressing that $\mathsf{k}(i_0)$ does not occur in t now instead requires that for each $(\mathcal{V}, \phi, \mathsf{k}(i)) \in \mathsf{ST}(t)$, the formula $\forall \mathcal{V}.\ \phi \Rightarrow i_0 \neq i$ holds. Taking all this into account, the EUF-CMA rule finally becomes:

$$\textsc{Euf-Cma}\ \frac{\mathcal{E}; \Theta; \Gamma \vdash \left(\bigwedge_{(\mathcal{V},\phi,\mathsf{k}(i)) \in \mathsf{ST}_{\mathsf{h}}(t,m)} \forall \mathcal{V}.\ \phi \Rightarrow i_0 \neq i \right) \wedge \left(\bigwedge_{(\mathcal{V},\phi,\mathsf{h}(u,\mathsf{k}(i))) \in \mathsf{ST}(t,m)} \forall \mathcal{V}.\ (\phi \wedge i_0 = i) \Rightarrow u \neq m \right)}{\mathcal{E}; \Theta; \Gamma \vdash \mathsf{h}(m, \mathsf{k}(i_0)) \neq t}$$

where $\mathsf{ST}_{\mathsf{h}}(t, m)$ is $\mathsf{ST}(t, m)$, where we excluded occurrences of k in subterms of the form $\mathsf{h}(\cdot, \mathsf{k}(\cdot))$. An additional premise (omitted here) checks that t and m are computable in polynomial time.

Making the verification effective. As presented here, the set of occurrences in a term may be infinite: thus, we cannot generate one premise for each occurrence. To alleviate this issue, we use heuristics that help Squirrel construct a proof obligation for the user that over-approximates this infinite set in a way that makes it expressible as a single formula. We illustrate how this can be done on an example.

Example 4.7. Continuing Example 4.6, consider arbitrary indices i_0, j_0 such that $j_0 < i_0$. Recall that $\mathsf{keys}(j_0)$ computes the list of keys $[\mathsf{k}(j_0), \ldots, \mathsf{k}(1)]$. In the style described above, a condition for a rule may require us to consider all elements $(\mathcal{V}, \phi, \mathsf{k}(i))$ of $\mathsf{ST}(\mathsf{att}(\mathsf{keys}(j_0)))$ (for any i) to show that i cannot be equal to i_0. There are countably many such elements, all of the form $(\emptyset, j_0 \neq 0\ \wedge\ j_0 - 1 \neq 0\ \wedge \cdots \wedge\ j_0 - \ell \neq 0, \mathsf{k}(j_0 - \ell))$ for $\ell \in \mathbb{N}$. They can all be subsumed by using instead the occurrence $(i, 0 < i \wedge i \leq j_0, \mathsf{k}(i))$. The corresponding proof obligation is $\forall i : \mathsf{index}.\ 0 < i \wedge i \leq j_0 \Rightarrow i \neq i_0$, which can indeed be proved under the hypothesis that $j_0 < i_0$.

4.5. Reasoning over traces: induction and case analysis

We now clarify the interpretation of traces in this logic, and what kind of reasoning we can perform over them. Following the modeling approach outlined in Section 3, recall that we may use $\mathsf{A} : \mathsf{index} \to \mathsf{timestamp}$ to represent an indexed action. Assuming for simplicity that we only have this action A, we restrict our attention to models $\mathbb{M}$ where

$$[\![\mathsf{timestamp}]\!]_{\mathbb{M}} = \{\mathsf{init}, \mathsf{undef}\} \cup \{\mathsf{A}(n) \mid n \in N\}$$

where N is a finite subset of $[\![\mathsf{index}]\!]_{\mathbb{M}}$ and undef is a special value used to interpret all timestamps that do not happen. Then, the constants init and A are interpreted naturally, independently of η, ρ:

$$[\![\mathsf{init}]\!]^{\eta,\rho}_{\mathbb{M}} = \mathsf{init} \qquad [\![\mathsf{A}]\!]^{\eta,\rho}_{\mathbb{M}}(n) = \mathsf{A}(n) \text{ for } n \in N \qquad [\![\mathsf{A}]\!]^{\eta,\rho}_{\mathbb{M}}(n) = \mathsf{undef} \text{ otherwise}$$

The following axiom is sound in this class of models:

$$\big[\forall x : \mathsf{timestamp}.\ \mathsf{happens}(x) \Rightarrow \big(x = \mathsf{init} \vee \exists i : \mathsf{index}.\ x = \mathsf{A}(i)\big)\big]$$

It can effectively be used to reason by case analysis on timestamps, but only at the level of local formulas. At the global level, it is tempting to postulate a stronger axiom:

$$\tilde{\forall} x : \mathsf{timestamp}.\ [\mathsf{happens}(x)] \mathrel{\tilde{\Rightarrow}} \big([x = \mathsf{init}] \mathrel{\tilde{\vee}} \tilde{\exists} i : \mathsf{index}.\ [x = \mathsf{A}(i)]\big)$$

However, this is too strong, and unsound in our class of models: consider for instance a random variable over timestamps that is init for some random tapes and some $\mathsf{A}(n)$ for other tapes. To avoid this problem, we crucially need to quantify over random variables that are actually constant, *i.e.* which depend neither on η nor on ρ. Assuming a global predicate $\mathsf{const}(\cdot)$ with exactly this meaning, we can now write the following, which is sound in our class of models:

$$\forall x : \mathsf{timestamp}.\ \mathsf{const}(x) \mathrel{\tilde{\wedge}} [\mathsf{happens}(x)] \mathrel{\tilde{\Rightarrow}} \big([x = \mathsf{init}] \mathrel{\tilde{\vee}} \tilde{\exists} i : \mathsf{index}.\ [x = \mathsf{A}(i)]\big)$$

In practice, such axioms are used in Squirrel proofs through the case tactic. This tactic, when given a timestamp, will perform a case analysis for local or global goals. However, when the goal is a global sequent, it will generate an additional subgoal to verify that the timestamp is constant, corresponding to the hypothesis in our axiom above. In practice, the constancy assumptions come from the target security property. For instance, the equivalence of protocols modeled in Section 3.2 would more precisely be stated as $\tilde{\forall}\tau.\ \mathsf{const}(\tau) \mathrel{\tilde{\Rightarrow}} \mathsf{frame}_{\mathsf{r}}@\tau \sim \mathsf{frame}_{\mathsf{i}}@\tau$.

Case analysis over timestamps is not always sufficient to prove a goal, and induction is sometimes needed. At the local level, induction is supported by the following axiom, which is obviously valid in our intended class of models:

$$[\forall (p : \mathsf{timestamp} \to \mathsf{bool}).\ (\forall x.\ (\forall y.\ y < x \Rightarrow p\ y) \Rightarrow p\ x) \Rightarrow (\forall x.\ p\ x)]$$

At the global level, the lack of higher-order quantification forces us to resort to an axiom scheme and, more importantly, we need $\mathsf{const}(\cdot)$ assumptions. For any global formula F, the following axiom is valid:

$$\big(\tilde{\forall} x.\ \mathsf{const}(x) \mathrel{\tilde{\Rightarrow}} (\tilde{\forall} y.\ \mathsf{const}(y) \mathrel{\tilde{\wedge}} [y < x] \mathrel{\tilde{\Rightarrow}} F\{x \mapsto y\}) \mathrel{\tilde{\Rightarrow}} F\big) \mathrel{\tilde{\Rightarrow}} \big(\tilde{\forall} x.\ \mathsf{const}(x) \mathrel{\tilde{\Rightarrow}} F\big)$$

Without the $\mathsf{const}(\cdot)$ restrictions, the axiom would be unsound: in a nutshell, the problem is that the well-founded relation $<$ over $[\![\mathsf{timestamp}]\!]_{\mathbb{M}}$ does not lift to a well-founded relation $[\cdot < \cdot]$ over random variables. This can be generalized to arbitrary sets of actions, and to arbitrary well-founded types: one may for instance reason by induction over natural numbers or trees, along the same lines.

4.6. Bi-deduction & cryptographic reductions

Given the scope of the logic, the question of automating some proof steps in Squirrel was quickly raised. A key property of computational indistinguishability is that it is preserved by public computations: if $\vec{u} \sim \vec{v}$ holds then for any adversarially computable function f, we have $f(\vec{u}) \sim f(\vec{v})$. This property, introduced in [Baelde et al. 2022], is of particular interest for two reasons:

— it is the theoretical justification behind many cryptographic proof steps;
— it is simple, notably because it is compositional, as opposed to indistinguishability $\sim$ which is not (see point 3 of Example 2.6).

For these reasons, this property turned-out to be a nice target for automation, reducing the proof-burden put on the user and allowing more high-level reasoning steps.

Bi-Deduction. Let $\sharp(\vec{v}_0; \vec{v}_1)$ be a pair of *sequences of terms*, representing the left and right scenarios of an equivalence. We say that it is *bi-deducible* from another pair $\sharp(\vec{u}_0; \vec{u}_1)$, which we write $\sharp(\vec{u}_0; \vec{u}_1) \rhd \sharp(\vec{v}_0; \vec{v}_1)$, if there exists a *simulator* function $\mathcal{S}$ that can be efficiently computed by the adversary and such that $\mathcal{S}([\![\vec{u}_i]\!]) = [\![\vec{v}_i]\!]$ for any $i \in \{0; 1\}$. Crucially, the same simulator $\mathcal{S}$ is used for both $i = 0$ and 1: $\mathcal{S}$'s behavior cannot depend on i as it does not know which side it is interacting with. The link between bi-deduction and indistinguishability is captured by the following rule:

$$\textsc{BiDeduce}\quad \frac{\sharp(\vec{u}_0;\vec{u}_1) \rhd \sharp(\vec{v}_0;\vec{v}_1) \qquad \vec{u}_0 \sim \vec{u}_1}{\vec{v}_0 \sim \vec{v}_1}$$

Indeed, any efficient adversary $\mathcal{A}$ against the conclusion of this rule can be turned into an adversary $\mathcal{B}$ against the premise by simply composing $\mathcal{A}$ with the simulator $\mathcal{S}$ witnessing the bi-deduction premises. $\mathcal{B}$ is an efficient adversary as both $\mathcal{A}$ and $\mathcal{S}$ are, and it has the same advantage as $\mathcal{A}$'s.

The BiDeduce rule transforms the $\forall$ quantification over adversary of an equivalence into an $\exists$ quantification: instead of proving a property against any possible adversaries, of which there are infinitely many, it suffices to establish the existence of a program witnessing bi-deduction. Of course, this observation is not new, as this is the key idea behind all reduction-based arguments, whether it be in cryptography, computability or computational complexity. The main novelty is how we use bi-deduction as an automated entailment checker integrated in Squirrel tactics.

Rules. As already mentioned, bi-deduction is compositional, which allows decomposing bi-deductions as follows:

$$\textsc{BD.Tuple}\quad \frac{\sharp(\vec{u}_0;\vec{u}_1) \rhd \sharp(v_1;w_1) \qquad \dots \qquad \sharp(\vec{u}_0;\vec{u}_1) \rhd \sharp(v_n;w_n)}{\sharp(\vec{u}_0;\vec{u}_1) \rhd \sharp(v_1,\dots,v_n;w_1,\dots,w_n)}$$

Here are two selected (simple) bi-deduction rules:

$$\textsc{BD.FA}\quad \frac{\sharp(\vec{u}_0;\vec{u}_1) \rhd \sharp(\vec{v}_0;\vec{v}_1)}{\sharp(\vec{u}_0;\vec{u}_1) \rhd f(\sharp(\vec{v}_0;\vec{v}_1))} \qquad \textsc{BD.Trans}\quad \frac{\sharp(\vec{u}_0;\vec{u}_1) \rhd \sharp(\vec{w}_0;\vec{w}_1) \qquad \sharp(\vec{u}_0,\vec{w}_0;\vec{u}_1,\vec{w}_1) \rhd \sharp(\vec{v}_0;\vec{v}_1)}{\sharp(\vec{u}_0;\vec{u}_1) \rhd \sharp(\vec{v}_0;\vec{v}_1)}$$

where common behaviors between the left and right scenarios are factorized when possible: *e.g.* $f(\sharp(\vec{v}_0;\vec{v}_1))$ denotes $\sharp(f(\vec{v}_0);f(\vec{v}_1))$. The BD.FA rule lets the adversary simulate the computation f: indeed, if $\mathcal{S}$ computes $\sharp(\vec{v}_0;\vec{v}_1)$ from $\sharp(\vec{u}_0;\vec{u}_1)$, then $f \circ \mathcal{S}$ computes $\sharp(f(\vec{v}_0);f(\vec{v}_1))$ from the same inputs. The transitivity rule BD.Trans allows extending the input $\sharp(\vec{u}_0;\vec{u}_1)$ of a simulator trying to compute $\sharp(\vec{v}_0;\vec{v}_1)$ by any sequence $\sharp(\vec{w}_0;\vec{w}_1)$ that is bi-deducible from $\sharp(\vec{u}_0;\vec{u}_1)$.

The more interesting bi-deduction rules deal with functions (seen as oracles) and conditioned terms, for which we now present two simplified rules. Conditioned terms are written $(t \mid \phi)$, which denotes the pair of terms $(\phi, \mathsf{if}\ \phi\ \mathsf{then}\ t\ \mathsf{else}\ \mathsf{witness})$, where $\mathsf{witness}$ is a function symbol of the same type as t.

$$\textsc{BD.Oracle}\quad \frac{\begin{array}{c} \rhd \sharp(t_0;t_1) \\ \vdash \begin{bmatrix} w_0\{y \mapsto t_0\} = v_0 \wedge \\ w_1\{y \mapsto t_1\} = v_1 \end{bmatrix} \end{array}}{\lambda y.\, \sharp(w_0;w_1) \rhd \sharp(v_0;v_1)} \qquad \textsc{BD.Cond}\quad \frac{\begin{array}{c} \rhd \sharp(\psi_0;\psi_1) \\ \vdash \begin{bmatrix} \psi_0 \Rightarrow (\phi_0 \wedge v_0 = w_0) \wedge \\ \psi_1 \Rightarrow (\phi_1 \wedge v_1 = w_1) \end{bmatrix} \end{array}}{\sharp((w_0 \mid \phi_0);(w_1 \mid \phi_1)) \rhd \sharp((v_0 \mid \psi_0);(v_1 \mid \psi_1))}$$

The BD.Oracle allows the simulator to call an oracle $\lambda y.\, \sharp(w_0;w_1)$ it received as input on any arguments $\sharp(t_0;t_1)$ that it can compute; the obtained terms $\sharp(w_0\{y \mapsto t_0\};w_1\{y \mapsto t_1\})$ must be established, in the second proof obligation, to be equal to the wanted terms $\sharp(v_0;v_1)$. In BD.Cond, the simulator retrieves from its inputs a conditioned term: the bi-deduction premise $\rhd \sharp(\psi_0;\psi_1)$ guarantees that the simulator knows when to extract the terms from its input; and the logical premise ensures that when it does (*i.e.* when

ψ_0 or ψ_1 holds), the extracted value $\sharp(w_0; w_1)$ is given as inputs (*i.e.* when ϕ_0 or ϕ_1 holds) and is equal to the wanted value $\sharp(v_0; v_1)$.

Implementation. We implemented a fully automated heuristic procedure for bi-deduction and integrated it in the `apply` tactic of Squirrel: concretely, it is used to automatically check the entailment between equivalences, *e.g.* $\vec{u}_0 \sim \vec{u}_1 \Rrightarrow \vec{v}_0 \sim \vec{v}_1$. While this procedure is heuristic, and thus incomplete, it remains a useful tool in practice.

Advanced bi-deduction. Bi-deduction gives a formal framework to establish the existence of a simulator $\mathcal{S}$ reducing an equivalence $\vec{u}_0 \sim \vec{u}_1$ to another equivalence $\vec{v}_0 \sim \vec{v}_1$. This looks very similar to the problem of applying a cryptographic assumption expressed as a pair of games $\mathcal{G}_0 \approx \mathcal{G}_1$ (in the style of Section 2): for example, given a target equivalence $\vec{u}_0 \sim \vec{u}_1$, does there exist $\mathcal{S}$ such that $\mathcal{S}^{\mathcal{G}_i}$ computes $\vec{u}_i$ for all $i \in \{0; 1\}$? Extending bi-deduction to be able to handle cryptographic games would allow supporting new cryptographic assumptions in Squirrel's logic and implementation without having to manually design rules as we did in Section 2.3 and Section 4.4. This would drastically reduce the theoretical and programming work needed to support a new rule while limiting the risk of mistakes. Furthermore, it would allow non-expert users to add new cryptographic assumptions in Squirrel simply by writing, in user-space, the pair of games $\mathcal{G}_0 \approx \mathcal{G}_1$ associated to the assumptions.

Unfortunately, the simulators covered by our previous notion of bi-deduction are too restricted for this. First, they can only interact with *stateless* oracles that are provided as first-order terms of the logic. Second, they are deterministic programs: while they may receive randomly sampled values from their inputs, they cannot do random samplings themselves. This is too limiting: oracles of cryptographic games are often stateful (*e.g.* EUF-CMA relies on a stateful log $\mathcal{L}$); and the simulators justifying Squirrel's cryptographic rules must sample all the randomness that does not come from the cryptographic game. Recently, we designed and implemented in Squirrel an extended notion of bi-deduction supporting stateful oracles and probabilistic simulators [Baelde et al. 2024]. This introduces a number of interesting aspects: we equipped the bi-deduction predicate with Hoare-style pre- and post-conditions to handle game statefulness; and random sampling usage is tracked through *randomness constraints* that ensure that the constructed simulator does not directly access the game's random samplings. This approach, which showed good initial results, may become the standard way of handling cryptographic assumptions in Squirrel.

5. CONCLUSION

We have developed a logic based on the CCSA logic described in [Bana and Comon-Lundh 2014], along with proof systems for verifying reachability and equivalence properties within that framework. Our work demonstrates that this approach offers a straightforward, high-level strategy for conducting computer-assisted proofs of cryptographic protocols, offering asymptotic security guarantees within the computational model. This is reinforced by the development of the interactive prover Squirrel and its application across different case studies [Baelde et al. 2021; Baelde et al. 2022; Cremers et al. 2022; Baelde et al. 2024], summarized in Figure 4. The number of LoC mentioned includes both the model and the proof script. The cryptographic assumptions on which the proof relies are also indicated, as well as the security properties under study. We analyze reachability properties, namely various forms of authentication, and some equivalence-based properties, *e.g.* unlinkability on several RFID protocols, an anonymity property, as well as a strong form of secrecy. The case studies are divided into two blocks, with non-stateful protocols first and then stateful ones. The Squirrel prover was extended to post-quantum security proofs in [Cremers et al. 2022], the corresponding protocols with security guarantees against a quantum attacker are indicated with a $\star$.

Protocol name	LoC	Assumptions	Security Properties
Basic Hash $\star$	100	PRF, EUF	auth. & unlinkability
Hash Lock $\star$	130	PRF, EUF	auth. & unlinkability
LAK (with pairs) $\star$	250	PRF, EUF	auth. & unlinkability
MW $\star$	300	PRF, EUF, XOR	auth. & unlinkability
Feldhofer $\star$	270	ENC-KP, INT-CTXT	auth. & unlinkability
Private authentication $\star$	100	CCA_1, ENC-KP	anonymity
Private authentication	100	CCA$	anonymity
Signed DDH [ISO 9798-3]	240	EUF, DDH	auth. & strong secrecy
$\mathrm{IKEV1}_{\mathrm{PSK}}$ $\star$	850	PRF	auth. & strong secrecy
$\mathrm{IKEV2}^{\mathrm{SIGN}}{}_{\mathrm{PSK}}$ $\star$	300	PRF, EUF	auth. & strong secrecy
BCGNP $\star$	355	PRF, CCA_1, XOR	strong secrecy
FSXY $\star$	620	PRF, CCA_1, XOR	strong secrecy
SC-AKE $\star$	745	PRF, EUF, CCA_1, XOR	auth. & strong secrecy
CANAuth	450	EUF	auth.
SLK06	80	EUF	auth.
YPLRK05	160	EUF	auth.
OSK (without reader)	320	ROM	strong secrecy
OSK	370	EUF	auth.
YubiKey	270	INT-CTXT	auth.
YubiHSM	720	CCA_1, INT-CTXT	auth.

Fig. 4. Some results obtained with the Squirrel prover on various protocols.

Ongoing and Future Work. As explained in Section 3, the Squirrel tool allows cryptographic protocols to be described in a process algebra, whereas the translation from protocols to the logic relies on the definition of mutually recursive functions modeling the protocol observables depending on the execution trace. To ease formal proofs, this translation does not follow the granularity of elementary execution steps in the process algebra. Instead, it groups together elementary steps in blocks following specific patterns. Whereas the soundness of the translation is easy to obtain for a simple class of protocols, some works remains to be done for more complex protocols.

We also aim to improve the automation of the prover. One option is to discharge parts of the proof that do not rely on any cryptographic assumption to SMT solvers. Another option is to rely on type systems: techniques based on typing have mainly been used in symbolic models, but we are currently working to develop such a system to obtain computational guarantees in the CCSA framework.

We aim to extend the scope of the post-quantum work initiated in [Cremers et al. 2022] and to analyze hybrid post-quantum protocols that rely on a combination of asymmetric post-quantum algorithms with well-known and well-studied pre-quantum asymmetric cryptography, *e.g.* based on the discrete logarithm problem.

Lastly, as mentioned in Section 3, the current notion of security in Squirrel differs from the standard one in the computational model, as the number of sessions we consider is arbitrary but fixed, in the sense that it does not depend on the security parameter. We plan to overcome this limitation, notably by leveraging the composition result outlined in [Comon et al. 2020]. Furthermore, we intend to expand on our methodology to go beyond asymptotic guarantees and provide concrete security bounds.

ACKNOWLEDGMENTS

This work received funding from the France 2030 program managed by the French National Research Agency under grant agreement No. ANR-22-PECY-0006.

REFERENCES

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.

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 *35th IEEE Computer Security Foundations Symposium, CSF 2022, Haifa, Israel, August 7-10, 2022*. IEEE, 289–304.

David Baelde, Adrien Koutsos, and Joseph Lallemand. 2023. A Higher-Order Indistinguishability Logic for Cryptographic Reasoning. In *38th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2023*. 1–13.

David Baelde, Adrien Koutsos, and Justine Sauvage. 2024. Cryptographic Reductions By Bi-Deduction. (March 2024). https://hal.science/hal-04511718

Gergei Bana and Hubert Comon-Lundh. 2012. Towards Unconditional Soundness: Computationally Complete Symbolic Attacker. In *Principles of Security and Trust - First International Conference, POST 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012, Proceedings (Lecture Notes in Computer Science)*, Pierpaolo Degano and Joshua D. Guttman (Eds.), Vol. 7215. Springer, 189–208.

Gergei Bana and Hubert Comon-Lundh. 2014. A Computationally Complete Symbolic Attacker for Equivalence Properties. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*. ACM, 609–620.

Manuel Barbosa, Gilles Barthe, Karthik Bhargavan, Bruno Blanchet, Cas Cremers, Kevin Liao, and Bryan Parno. 2021. SoK: Computer-Aided Cryptography. In *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*. IEEE, 777–795.

Gilles Barthe, Benjamin Grégoire, and Santiago Zanella Béguelin. 2009. Formal certification of code-based cryptographic proofs. In *Proceedings of the 36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2009, Savannah, GA, USA, January 21-23, 2009*, Zhong Shao and Benjamin C. Pierce (Eds.). ACM, 90–101.

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)*, Phillip Rogaway (Ed.), Vol. 6841. Springer, 71–90.

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.

Bruno Blanchet. 2001. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *14th IEEE Computer Security Foundations Workshop (CSFW-14 2001), 11-13 June 2001, Cape Breton, Nova Scotia, Canada*. IEEE Computer Society, 82–96.

Bruno Blanchet. 2008. A Computationally Sound Mechanized Prover for Security Protocols. *IEEE Trans. Dependable Secur. Comput.* 5, 4 (2008), 193–207.

Mayla Brusò, Konstantinos Chatzikokolakis, and Jerry den Hartog. 2010. Formal Verification of Privacy for RFID Systems. In *23rd IEEE Computer Security Foundations Symposium, CSF 2010, Edinburgh, United Kingdom, July 17-19, 2010*. IEEE Computer Society, 75–88.

Hubert Comon, Charlie Jacomme, and Guillaume Scerri. 2020. Oracle Simulation: A Technique for Protocol Composition with Long Term Shared Secrets. In *2020 ACM SIGSAC Conference on Computer and Communications Security, CCS 2020*. ACM, 1427–1444.

Cas Cremers, Caroline Fontaine, and Charlie Jacomme. 2022. A Logic and an Interactive Prover for the Computational Post-Quantum Security of Protocols. In *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022*. IEEE, 125–141.

Danny Dolev and Andrew Chi-Chih Yao. 1981. On the Security of Public Key Protocols (Extended Abstract). In *22nd Annual Symposium on Foundations of Computer Science, Nashville, Tennessee, USA, 28-30 October 1981*. IEEE Computer Society, 350–357.

Simon Meier, Benedikt Schmidt, Cas Cremers, and David A. Basin. 2013. The TAMARIN Prover for the Symbolic Analysis of Security Protocols. In *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings (Lecture Notes in Computer Science)*, Natasha Sharygina and Helmut Veith (Eds.), Vol. 8044. Springer, 696–701.

Adam Petcher and Greg Morrisett. 2015. The Foundational Cryptography Framework. In *Principles of Security and Trust - 4th International Conference, POST 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015, Proceedings (Lecture Notes in Computer Science)*, Riccardo Focardi and Andrew C. Myers (Eds.), Vol. 9036. Springer, 53–72.

Nikhil Swamy, Catalin Hritcu, Chantal Keller, Aseem Rastogi, Antoine Delignat-Lavaud, Simon Forest, Karthikeyan Bhargavan, Cédric Fournet, Pierre-Yves Strub, Markulf Kohlweiss, Jean Karim Zinzindohoue, and Santiago Zanella Béguelin. 2016. Dependent types and multi-monadic effects in F. In *43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016*, Rastislav Bodík and Rupak Majumdar (Eds.). ACM, 256–270.