An Interactive Prover for Protocol Verification in the Computational Model

David Baelde*, Stéphanie Delaune†, Charlie Jacomme‡, Adrien Koutsos§ and Solène Moreau†
* LMF, ENS Paris-Saclay & CNRS, Université Paris-Saclay, France
†Univ Rennes, CNRS, IRISA, France
‡CISPA Helmholtz Center for Information Security, Germany
§Inria Paris, France

***Abstract*—Given the central importance of designing secure protocols, providing solid mathematical foundations and computer-assisted methods to attest for their correctness is becoming crucial. Here, we elaborate on the formal approach introduced by Bana and Comon in [10], [11], which was originally designed to analyze protocols for a fixed number of sessions, and lacks support for proof mechanization.**

In this paper, we present a framework and an interactive prover allowing to mechanize proofs of security protocols for an arbitrary number of sessions in the computational model. More specifically, we develop a meta-logic as well as a proof system for deriving security properties. Proofs in our system only deal with high-level, symbolic representations of protocol executions, similar to proofs in the symbolic model, but providing security guarantees at the computational level. We have implemented our approach within a new interactive prover, the SQUIRREL prover, taking as input protocols specified in the applied pi-calculus, and we have performed a number of case studies covering a variety of primitives (hashes, encryption, signatures, Diffie-Hellman exponentiation) and security properties (authentication, strong secrecy, unlinkability).

***Index Terms*—Security Protocols, Formal Methods, Observational Equivalence, Computational Security, Interactive Prover.**

I. INTRODUCTION

Given the importance and difficulty of designing secure communication protocols, computer scientists have strived to provide solid mathematical foundations, formal methods and tools for the computer-aided verification of security protocols.

A successful approach in this line of research is to model cryptographic messages as formal terms subject to some equational theory representing attacker capabilities. Originally proposed by Dolev and Yao [36], this idea has been refined over the years, resulting in a variety of so-called *symbolic* models, based, e.g., on the applied pi-calculus [2] or multiset rewriting [46]. These models capture large classes of attackers and allow the automated verification of protocols. This has lead to tools such as PROVERIF [20], TAMARIN [45], DEEPSEC [26] and others [5], [25], [32], [37], which have been used to prove the security of widely deployed protocols, e.g. [14], [18], [35].

To discuss the general principle and limitations of the symbolic model, let us consider a simple protocol where a tag (role T) with identity id carrying its own hashing key k is authenticated by a reader (role R) that has access to a database of legitimate pairs $\langle id, k\rangle$:

$$\begin{array}{rcl} R \to T & : & n \\ T \to R & : & id \oplus \mathsf{H}(n,k) \end{array}$$

In words, the reader sends a random number n and, when receiving a reply, verifies that there is some $\langle id, k\rangle$ in its database such that the reply is the identity id XORed with the hash of n using the key k. To verify that an attacker cannot impersonate a tag with data $\langle id_0, k_0\rangle$, we need to check that the attacker cannot derive a message that the reader would accept, whatever knowledge they may have obtained from previous interactions. Assuming for simplicity two identities id_0 and id_1, and that the attacker is only active during its last interaction, we have to verify that, for distinct names $(n_j)_{0\leq j\leq p}$, and for any $i_1,\ldots,i_p \in \{0,1\}$, there is *no* context C made of public function symbols such that:

$$C[n_j, id_{i_j} \oplus \mathsf{H}(n_j, k_{i_j})]_{1\leq j\leq p} \;=_{\mathsf{E}}\; id_0 \oplus \mathsf{H}(n_0, k_0)$$

where $=_{\mathsf{E}}$ is equality modulo a relevant equational theory.

Rephrased like this, the problem may be solved using rewriting techniques such as unification, itself based on completion or variant computation [30], [48]. However, the equational theory E would need to contain equations reflecting the algebraic properties of XOR which are typically problematic for these techniques. In fact, PROVERIF and DEEPSEC do not support XOR, and TAMARIN only provides limited support for it. Supporting primitives with rich algebraic properties (e.g., blind signatures, exponentiation, XOR) is probably the main challenge that these tools are currently facing.

An obvious limitation of the symbolic model is that it can only be used to find *logical* flaws in protocols, e.g. man-in-the-middle [44] or reflection attacks [15]. Indeed, security in the symbolic model is weaker than the cryptographer's standard security notion, based on a *computational* model where adversaries are arbitrary probabilistic polynomial-time (PTIME) Turing machines. As an illustration, nonces are modelled in the computational model as long bitstrings that are drawn uniformly at random. Two nonces correspond to the same probability distribution, and are thus indistinguishable,

The research leading to these results has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No 714955-POPSTAR), and from the French National Research Agency (ANR) under the project TECAP.

but the probability that two distinct nonces take the same value is negligible. The attacker may guess a bit of a nonce but has a negligible probability of guessing the whole nonce. In the symbolic model, distinct names are indistinguishable and are not equal modulo E, but the partial guessing of a nonce cannot be modelled: a name is either known or unknown to the attacker. Significant research efforts have been done to get the best of both worlds through the development of *computational soundness* results, but unfortunately they only apply under strong hypotheses, and have modularity issues [3], [33], [34].

In [10], [11], Bana and Comon have proposed a new approach to security proofs, which they call *computationally complete symbolic attacker* (CCSA). It relies on the symbolic setting of first-order logic, but avoids the limitations of the symbolic models mentioned above. Instead of modelling attacker capabilities using rules stating what the adversary *can* do, their method relies on the specification of what the attacker *cannot* do. Starting from the security properties of cryptographic primitives, they derive some axioms expressing which pairs of sequences of messages are indistinguishable. They show that their axioms are sound w.r.t. the interpretation of terms as probabilistic PTIME Turing machines. Therefore, a proof of a security property from these axioms implies security in the computational model under the initial cryptographic assumptions. This approach has been demonstrated on various protocols to obtain formal proofs of security [8], [9], [28], [41], [49]. Going back to our example, authentication would be expressed more abstractly as:

$$\mathsf{EQ}(\mathsf{att}(\mathit{frame}), id_0 \oplus \mathsf{H}(n_0, k_0)) \sim \mathsf{false}, \text{ where}$$
$$\mathit{frame} \overset{\text{def}}{=} \langle n_1, id_{i_1} \oplus \mathsf{H}(n_1, k_{i_1}), n_2, id_{i_2} \oplus \mathsf{H}(n_2, k_{i_2}), \ldots \rangle.$$

Here, the binary function symbol EQ is interpreted as bitstring equality, and $\sim$ as indistinguishability. The function symbol att stands for an arbitrary probabilistic PTIME computation performed by the attacker, taking past messages as input.

Intuitively, the overall statement expresses that there is a negligible probability that $\mathsf{att}(\mathit{frame})$ yields a message that the reader would accept. It can be proved using just two axioms: first, we use the properties of XOR to obtain an equality between $\mathsf{H}(n_0, k_0)$ and $id_0 \oplus \mathsf{att}(\mathit{frame})$; second, assuming that the keyed hash function satisfies existential unforgeability under chosen message attack (EUF-CMA), we conclude that this equality is false with overwhelming probability given that the message n_0 is fresh and has thus not been hashed in frame.

In contrast with the treatment of our example in the symbolic model, the CCSA approach yields a proof that is immediately relevant in the standard model of probabilistic PTIME machines and which relies on explicit and standard cryptographic assumptions. Finally, applying the axioms did not require complex rewriting techniques: we only had to verify a simple equality involving XOR, and never had to consider all possible contexts as in the symbolic model.

Two problems prevent a more widespread use of the CCSA approach. First, it is limited to bounded executions: given a protocol and a bound on its execution traces, one can derive a series of indistinguishability goals that need to be (separately) proved to guarantee the security of the protocol. Second, proofs are manual: proving a non-trivial goal in detail is tedious; proving all the goals resulting from all possible executions is not manageable. Works on decision procedures remain of limited use so far, as they are limited to trace properties [29] (e.g. secrecy, authentication) or a very restrictive set of axioms [42].

Contributions: In this paper, we elaborate on the CCSA approach to solve these two problems. Our first contribution, presented in Sections III and IV, is a *meta-logic* over the base logic of Bana and Comon [11]. In the base logic, one has to separately prove a family of indistinguishability goals, one for each possible protocol execution within a given bound. Formulas of our meta-logic express properties of all execution traces of a protocol, which allows to capture the family of base logic goals as a single formula. Security properties expressed in our meta-logic have the standard computational meaning, and provide guarantees for an arbitrary number of sessions that does not depend on the security parameter (this is discussed in detail in Section IV-C).

We then design proof systems for deriving security properties expressed as meta-logic formulas, in part by lifting base logic axioms. We consider both trace properties, which are useful to model authentication properties, and equivalence properties, which allow to express privacy properties such as anonymity and unlinkability. Our meta-logic enables useful interaction between the two kinds of properties. For instance, in several of our case studies, we establish unlinkability by proving *en passant* an authentication property. This second contribution is developed in Sections V and VI.

Our third contribution is the implementation of the interactive prover SQUIRREL [50], which allows to reason on protocols specified in an applied pi-calculus using our framework. Thanks to basic automation techniques coming from first-order logic, we have been able to carry out several case studies using this tool. These results, presented in Section VII, cover a variety of primitives (hashes, signatures, Diffie-Hellman exponentiation, encryption) and security properties (authentication, strong secrecy, unlinkability). Although this is not our primary goal, some of the proofs obtained here are firsts.

Related Work: We have already discussed the limitations that are inherent to symbolic models, and now focus on tools which provide guarantees in the computational model. Several such systems exist, based on different approaches. For instance, CRYPTOVERIF [21] proofs are based on high-level game transformations, EASYCRYPT [13] is built on a general-purpose probabilistic relational Hoare logic which can be used to formalize most pen-and-paper cryptographic proofs, and CRYPTHOL [17] goes even further by embedding the computational model in the proof assistant Isabelle/HOL. Finally, F* [19] is a general-purpose program verification framework which can be used (via external arguments) to provide computational security guarantees. The various approaches can be compared on several criteria [12]; we mention a few to highlight differences with our tool.

Like CryptoVerif, our protocol specifications are given in the applied pi-calculus, although our language is less detailed and does not provide a link with implementations. The strongest tools for verifying implementations remain EasyCrypt and, chiefly, F*.

Unlike EasyCrypt and CryptoVerif, we only provide asymptotic security bounds. Our approach hides from the user all quantitative aspects such as probabilities and the security parameter and, on the surface, our tool is as simple as symbolic verification frameworks. In contrast, EasyCrypt users often have to carry out probabilistic reasoning, and come up with complex security bounds, which can result in long proofs. Such aspects are automated in CryptoVerif. In general, the current level of automation of our tool sits somewhere between EasyCrypt and CryptoVerif.

The most important difference between our tool and earlier ones is the associated proof methodology: CryptoVerif relies on game transformations and EasyCrypt performs Hoare-style proofs of programs, while we reason over execution traces of protocols. Our proofs are akin to Tamarin's backward reachability analysis, driven e.g. by our unforgeability tactic. We give in Appendix E an in-depth comparison with EasyCrypt and CryptoVerif, based on the analysis of the Basic Hash protocol in the three tools.

II. Overview

In this section, we give an overview of our framework and tool, using as a running example the Basic Hash protocol. The Squirrel prover and our case studies can be found in [50].

Example 1. *We consider the Basic Hash protocol as described in [23], which is an RFID protocol involving multiple tags and readers. Each tag stores a secret key that is never updated, and the readers have access to a shared database containing all these keys. The protocol is as follows:*

$$T \rightarrow R : \langle n, \mathsf{H}(n, key)\rangle.$$

Here, n is a fresh name and key is the secret key. When receiving a message, the reader checks that it is a pair whose second component is a hash of the first component using one of the keys from the database.

```
hash H
abstract ok : message
abstract error : message
name key : index → message
channel c
process tag(i,j:index) =
  new n; out(c, ⟨n, H(n,key[i])⟩)
process reader(j:index) =
  in(c,x);
  try find i such that snd(x)=H(fst(x),key[i])
  in out(c,ok)
  else out(c,error)

system (!_j R: reader(j) | !_i !_j T: tag(i,j)).
```

Listing 1. Basic Hash protocol in Squirrel

Listing 1 shows a formal description of the Basic Hash protocol written in the input language of our tool, which is close to the classical applied pi-calculus. More specifically, we consider a scenario featuring several reader sessions with access to the database, and several tags where each tag can play multiple sessions. The try find instruction encodes the database lookup performed by the reader: the then branch is executed with some value of i for which the required condition holds, if any such i exists, otherwise the else branch is executed.

We now describe informally how to instantiate our framework to analyze this protocol — in practice, our tool performs this instantiation automatically from the applied pi-calculus specification. We consider a set of *actions* representing each step of the protocol: $\mathsf{T}[i,j]$ is the action performed by the j^{th} session of tag i, $\mathsf{R}[j,i]$ represents the action of a reader session j when it has found a value i for which the second component of its input is a hash of its first component with $\mathsf{key}[i]$, and $\mathsf{R1}[j]$ represents the action of a reader session j when no such i can be found.

Using the user syntax, we now express an authentication property on the Basic Hash protocol.

```
goal auth :
  forall (i:index, j:index),
    cond@R(j,i) ⇒
    exists (j':index), T(i,j') < R(j,i)
    && fst(input@R(j,i)) = fst(output@T(i,j'))
    && snd(input@R(j,i)) = snd(output@T(i,j')).
```

Listing 2. Reachability goal in Squirrel

Here $\mathsf{cond@R}[j,i]$ is a *macro* which stands for the executability condition of action $\mathsf{R}[j,i]$, where the reader recognizes a valid input message w.r.t. some key $\mathsf{key}[i]$. Our authentication goal expresses that, whenever this condition holds, there must be some session j' of tag i (the one using $\mathsf{key}[i]$) that has been executed before $\mathsf{R}[j,i]$. Moreover, the output of the tag's action coincides with the input of the reader's action. We may note that we express this correspondence on each projection. Indeed, for some implementations of the pairing primitive, the equality of projections does not imply the equality of pairs.

This authentication goal can be proved in our tool using a succession of four *tactics*:

```
simpl. expand cond@R(j,i). euf M0. exists j1.
```

The first tactic simply introduces variables i and j and the assumption $\mathsf{cond@R}[j,i]$ identified by M0. The second tactic expands this macro into its meaning, i.e.

$$\mathsf{snd}(\mathsf{input@R}[j,i]) = \mathsf{H}(\mathsf{fst}(\mathsf{input@R}[j,i]), \mathsf{key}[i]).$$

We then use the EUF-CMA assumption: the condition states that if $\mathsf{snd}(\mathsf{input@R}[j,i])$ is a valid hash of $\mathsf{fst}(\mathsf{input@R}[j,i])$, thus the term $\mathsf{fst}(\mathsf{input@R}[j,i])$ must be equal to a message that has previously been hashed, i.e. some m such that $\mathsf{H}(m, \mathsf{key}[i])$ appears in $\mathsf{snd}(\mathsf{input@R}[j,i])$ or $\mathsf{fst}(\mathsf{input@R}[j,i])$. Actually, $\mathsf{input@R}[j,i]$ refers to all messages outputted so far in the execution, and the only hashed messages outputted by the protocol with the key $\mathsf{key}[i]$ are the names $\mathsf{n}[i,j]$ (note that n is parametrized by i and j since it has been generated below the replications indexed by i and j). Therefore we deduce that there exists a tag's session j_1 occurring before the action $\mathsf{R}[j,i]$ such that $\mathsf{n}[i,j_1] = \mathsf{fst}(\mathsf{input}@\mathsf{R}[j,i])$. We conclude by instantiating the existential quantification over j' by j_1.

Our framework does not only provide a proof system for trace properties, but also allows to prove equivalence properties. For illustration purposes, let us consider the following unlinkability property for the Basic Hash protocol: we want to prove that a scenario where a tag with identity i can play many sessions is indistinguishable to a scenario where each tag with identity i can play only one session. To this end, we make use of bi-processes with diff terms (as done e.g. in ProVerif [22] or Tamarin [16]), and we replace in the protocol specification given in Listing 1 every $\mathsf{key}[i]$ with $\mathsf{diff}(\mathsf{key}[i], \mathsf{key}'[i,j])$. On the left side of the bi-process, the key is the same for all sessions of the tag with identity i, whereas on the right side each session j of the tag with identity i uses a new key.

We can then prove that the two projections of this bi-process are observationally equivalent. The proof proceeds by induction: we show that indistinguishability holds between the left and right-hand side's frames at any point of any trace, assuming that the same holds for the previous points. We then consider the three possible actions of the processes. For an action $\mathsf{T}[i,j]$, we use the PRF assumption on the hash function to replace the hashes of the fresh messages $\mathsf{H}(\mathsf{n}[i,j],\ldots)$ by fresh names; we can then conclude since indistinguishability is preserved by the addition of fresh names on both sides. For a reader's action $\mathsf{R}[j,i]$, we must show the equivalence (up-to negligible probability) between the conditions of the action for the two projections of our bi-process. This is an immediate consequence of the previous authentication property (and its obvious converse) which holds for our two projections. The case of actions $\mathsf{R1}[j]$ is handled similarly.

III. Modelling protocols – Syntax

In this section, we introduce the syntax of our meta-logic, which is an extension of the base logic of [11] with timestamps, indices and macros, before describing how to use it to model protocols. Along this section, we illustrate our notions using the Basic Hash protocol introduced in Section II.

A. Meta-Logic

Syntactically, our meta-logic is a many-sorted first-order logic. Terms of the meta-logic (*meta-terms*) are of three possible sorts:

- terms of sort *message* represent bitstrings manipulated and exchanged by protocol's participants;
- terms of sort *timestamp* represent time points in an execution trace of a protocol;
- terms of sort *index* are used to identify unbounded collections of objects, e.g. sessions of a role or items in a database.

$$
\begin{array}{rcl}
T & := & \tau \mid \mathsf{init} \mid \mathsf{a}[i_1,\ldots,i_k] \mid \mathsf{pred}(T) \\
t & := & x \mid \mathsf{n}[i_1,\ldots,i_k] \mid \mathsf{f}[i_1,\ldots,i_k](t_1,\ldots,t_n) \\
 & \mid & \mathsf{input}@T \mid \mathsf{output}@T \mid \mathsf{frame}@T \\
 & \mid & \mathsf{if}\ \phi\ \mathsf{then}\ t\ \mathsf{else}\ t' \\
 & \mid & \mathsf{find}\ \vec{i}\ \mathsf{suchthat}\ \phi\ \mathsf{in}\ t\ \mathsf{else}\ t' \\
A & := & t = t' \mid i = i' \\
 & \mid & T = T' \mid T \leq T' \mid \mathsf{cond}@T \mid \mathsf{exec}@T \\
\phi & := & A \mid \top \mid \bot \mid \phi \wedge \phi' \mid \phi \vee \phi' \mid \phi \Rightarrow \phi' \mid \neg\phi \\
 & \mid & \forall i.\phi \mid \exists i.\phi \mid \forall \tau.\phi \mid \exists \tau.\phi
\end{array}
$$

Fig. 1. Syntax of meta-terms and meta-formulas

We assume three infinite sets of variables: $\mathcal{X}$ (whose elements are noted x, y, z) for message variables; $\mathcal{I}$ (whose elements are noted i, j) for index variables; $\mathcal{T}$ (whose elements are noted τ) for timestamp variables.

We assume a set $\mathcal{F}$ of indexed function symbols (used to model encryptions, pairs, …). Each of these symbols comes with an *index arity* as well as *a message arity*: if $\mathsf{f} \in \mathcal{F}$ has index arity k and message arity n, then for all index variables $i_1,\ldots,i_k$ and meta-terms $t_1,\ldots,t_n$, we have that $\mathsf{f}[i_1,\ldots,i_k](t_1,\ldots,t_n)$ is a meta-term.

Example 2. *Function symbols representing cryptographic primitives will have index arity* 0*, and a message arity depending on the kind of primitive. For instance, we use* H *of message arity* 2 *to model a keyed hash,* $\langle\cdot,\cdot\rangle$ *of message arity* 2 *to model concatenation, and* fst *(resp.* snd*) of message arity* 1 *to model the first (resp. second) projection. Function symbols representing identities (for example, a constant value associated to each tag) have* 0 *as message arity and* 1 *(or even more) as index arity.*

We assume a set $\mathcal{N}$ of indexed name symbols (modelling random samplings of length η, the security parameter) and a set of indexed action symbols $\mathcal{A}$ (modelling specific timestamps). These indexed symbols only have an index arity: they cannot be applied to meta-terms.

Example 3. *To model the Basic Hash protocol in our framework, we consider two names* $\mathsf{key}, \mathsf{n} \in \mathcal{N}$*:* key *has index arity* 1*, and* $\mathsf{key}[i]$ *models the key associated to the tag* i*;* n *has index arity* 2*, and* $\mathsf{n}[i,j]$ *represents the name used by the session* j *of the tag* i*. Regarding actions symbols we let* $\mathcal{A}$ *be the set of three indexed action symbols:* a_T *and* a_R *of index arity 2 and* a_{R1} *of index arity 1. These action symbols correspond to what is called* $\mathsf{T}, \mathsf{R}, \mathsf{R1}$ *in Section II.*

Definition 1. *Given a meta-logic signature* $\Sigma = (\mathcal{F}, \mathcal{N}, \mathcal{A})$ *and some sets of variables* $\mathcal{X}$*,* $\mathcal{I}$ *and* $\mathcal{T}$*, we give in Fig. 1 the syntax of meta-terms of sort message (noted* t*) and timestamp (noted* T*), and the syntax of meta-formulas (noted* ϕ*). The only meta-terms of sort index are index variables.*

For any meta-term t*, we let* $\mathsf{st}(t)$ *and* $\mathsf{fv}(t)$ *be, resp., the set* *of subterms of t and the free variables of t of any sort.*

Note that meta-terms and meta-formulas are mutually inductive, due to conditional and lookup constructs in terms. Lookups generalize conditionals: $\mathsf{find}\ \vec{i}\ \mathsf{suchthat}\ \phi\ \mathsf{in}\ t\ \mathsf{else}\ t'$ evaluates to t where indices $\vec{i}$ are bound to values such that ϕ holds if such values exist, and t' otherwise. Again, this lookup construct can be computed thanks to the finiteness of the domain of interpretation of indices. The special timestamp constant init stands for the initial time point.

A key ingredient of our meta-logic are macros, which are used to refer to protocol executions. We have message macros $\mathsf{input}@T$ and $\mathsf{output}@T$ to refer to the input and output messages of the action executed at time T. The macro $\mathsf{frame}@T$ represents all the information available to the attacker at that time: essentially, it is the sequence of all output messages from the start of the protocol until time T. We also have boolean macros $\mathsf{cond}@T$ and $\mathsf{exec}@T$ which respectively encode the execution condition of the action at time T and the conjunction of all such conditions until time T.

B. Protocols as Sets of Actions

We model a protocol as a finite set of actions. Each action represents a basic step of the protocol where the attacker provides an input, a condition is checked, and finally an output is emitted. Formally, an action is defined by an action symbol identifying it, and an action description giving its semantics. Actions are indexed, allowing for unbounded executions.

Definition 2. *An* action $\mathsf{a}[i_1,\dots,i_k].(\phi,o)$ *is formed from an action symbol* a *of index arity k, distinct indices $i_1,\dots,i_k$, a meta-logic formula ϕ and a meta-logic term o of sort message such that* $\mathsf{fv}(\phi,o)\subseteq\{i_1,\dots,i_k\}$*. The formula ϕ is the* condition *of the action, and o its* output.

An action $\mathsf{a}[i_1,\dots,i_k].(\phi,o)$ models that o will be emitted provided that ϕ holds, but does not specify a failure case. Conditional branching may be modelled using two actions: one with the condition for the positive branch, and one with the negation of the condition for the negative branch. Alternatively, a single action with a trivial condition may be used, and an output term that performs the conditional branching. As we shall see, actions are chosen by the attacker, hence the second option gives less power to the attacker.

A protocol is a set of actions equipped with a dependency relation, which constrains the order of execution of actions.

Definition 3. *Given a finite set $\mathcal{A}$ of action symbols, a protocol $\mathcal{P}=(\mathcal{P}_\mathcal{A},<)$ over $\mathcal{A}$ is a finite set $\mathcal{P}_\mathcal{A}$ of actions, one for each action symbol, equipped with a partial order $<$ over terms of the form $\mathsf{a}[\vec{i}]$ with $\mathsf{a}\in\mathcal{A}$. We require that:*

- *$<$ is insensitive to the choice of specific index variables: $\mathsf{a}[i_1,\dots,i_k] < \mathsf{a}'[j_1,\dots,j_{k'}]$ iff $\mathsf{a}[\sigma(i_1),\dots,\sigma(i_k)] < \mathsf{a}'[\sigma(j_1),\dots,\sigma(j_{k'})]$ for any $\mathsf{a},\mathsf{a}',\vec{i}$ and $\vec{j}$ and for any bijective variable renaming $\sigma:\mathcal{I}\to\mathcal{I}$;*
- *actions only refer to information about previously executed actions; for every $\mathsf{a}[\vec{i}].(\phi,o)\in\mathcal{P}_\mathcal{A}$, each subterm of ϕ and o of sort timestamp:*
  - *(i) either appears in an input macro $\mathsf{input}@\mathsf{a}[\vec{i}]$,*
  - *(ii) or is of the form $\mathsf{a}'[\vec{j}]$ where $\mathsf{a}'[\vec{j}] < \mathsf{a}[\vec{i}]$.*

Intuitively, an action can refer to its own input, and can otherwise only refer to timestamps corresponding to actions that occur strictly before it. We derive $\leq$ from $<$ as usual: $\alpha\leq\beta$ when $\alpha<\beta$ or $\alpha=\beta$.

Example 4. *For illustration purposes, we consider a protocol made of two actions*

$$\mathsf{a}[i].(\top,\mathsf{ok}) \quad \textit{and} \quad \mathsf{b}[i].(\top,\langle\mathsf{input}@\mathsf{a}[i],\mathsf{input}@\mathsf{b}[i]\rangle)$$

with $\mathsf{a}[i]<\mathsf{b}[i]$. Intuitively, this corresponds to multiple sessions indexed by i, where each session inputs an arbitrary message, outputs ok*, then inputs another message before outputting the pair of the two messages it has received. Note that action $\mathsf{b}[i]$ is well-formed because $\mathsf{a}[i]<\mathsf{b}[i]$ and $\mathsf{b}[i]$ only occurs in the action's components through* $\mathsf{input}@\mathsf{b}[i]$.

Example 5. *We model the Basic Hash protocol in our framework using a set of three actions with an empty dependency relation:*

$$\mathsf{a}_T[i,j].\big(\top,\langle\mathsf{n}[i,j],\mathsf{H}(\mathsf{n}[i,j],\mathsf{key}[i])\rangle\big)$$

$$\begin{array}{l}\mathsf{a}_R[j,i].\big(\\ \mathsf{snd}(\mathsf{input}@\mathsf{a}_R[j,i])=\mathsf{H}(\mathsf{fst}(\mathsf{input}@\mathsf{a}_R[j,i]),\mathsf{key}[i]),\\ \mathsf{ok}\big)\end{array}$$

$$\begin{array}{l}\mathsf{a}_{R1}[j].\big(\\ \forall i.\ \mathsf{snd}(\mathsf{input}@\mathsf{a}_{R1}[j])\neq\mathsf{H}(\mathsf{fst}(\mathsf{input}@\mathsf{a}_{R1}[j]),\mathsf{key}[i])\big),\\ \mathsf{error}\big)\end{array}$$

We can instantiate the indices of an action by concrete values to yield *concrete actions*, which represent distinct copies of the original action.

Definition 4. *Given a set $\mathcal{A}$ of action symbols, a* concrete action *is an action symbol $\mathsf{a}\in\mathcal{A}$ applied to k integers (where k is the index arity of a). The partial order of a protocol $\mathcal{P}=(\mathcal{P}_\mathcal{A},<)$ is lifted to concrete actions in the natural way: for any mapping $\sigma:\mathcal{I}\to\mathbb{N}$, $\mathsf{a}[\sigma(i_1),\dots,\sigma(i_k)] < \mathsf{b}[\sigma(j_1),\dots,\sigma(j_l)]$ holds when $\mathsf{a}[i_1,\dots,i_k]<\mathsf{b}[j_1,\dots,j_l]$.*

Finally, for a given protocol, we can consider its possible *interleavings*, e.g. the possible sequences of actions that are compatible with its dependency relation.

Definition 5. *Given a protocol $\mathcal{P}=(\mathcal{P}_\mathcal{A},<)$, an* interleaving *is a sequence of concrete actions $\alpha_1\dots\alpha_n$ in which no concrete action occurs twice, and such that, for every $1\leq i\leq n$, for every concrete action β such that $\beta<\alpha_i$, there exists $1\leq j<i$ such that $\beta=\alpha_j$.*

The constraints on interleavings are necessary but insufficient conditions for a sequence of concrete actions to be executable. The actual executability of an interleaving will be a probabilistic notion, and will depend on the implementation of cryptographic primitives. It is addressed in the next section.

Example 6. *Going back to the simple protocol introduced in Example 4, we have concrete actions* $\mathsf{a}[1]$, $\mathsf{b}[1]$, $\mathsf{a}[2]$,

$\mathsf{b}[2]$, ... with $\mathsf{a}[1] < \mathsf{b}[1]$, $\mathsf{a}[2] < \mathsf{b}[2]$, ... indicating that, in each session, b has to occur after a. The sequence $\mathsf{a}[2]\,\mathsf{a}[1]\,\mathsf{b}[1]\,\mathsf{b}[2]$ is an interleaving, but $\mathsf{a}[2]\,\mathsf{a}[1]\,\mathsf{b}[1]\,\mathsf{b}[2]\,\mathsf{a}[2]$, and $\mathsf{b}[2]\,\mathsf{a}[1]\,\mathsf{b}[1]\,\mathsf{a}[2]$ are not.

Note that our notion of dependency ordering can be used to impose phase constraints: declaring $\mathsf{a}[i] < \mathsf{b}[j]$ imposes that, in all interleavings, any concrete actions $\mathsf{a}[k_1]$ is executed before all concrete actions $\mathsf{b}[k_2]$. In the tool, such extra constraints can be specified with an axiom, as follows:

```
axiom phase: forall(i,j:index), a(i) < b(j).
```

Example 7. *Continuing our running example, $\mathsf{a}_T[1,1]$, $\mathsf{a}_T[1,2]$, $\mathsf{a}_T[2,1]$, $\mathsf{a}_R[3,1]$, and $\mathsf{a}_{R1}[3]$ are concrete actions with no dependency. Some possible interleavings are:*

1) $\mathsf{a}_T[1,2]\,\mathsf{a}_R[3,1]$;
2) $\mathsf{a}_T[1,2]\,\mathsf{a}_R[3,1]\,\mathsf{a}_{R1}[3]$;
3) $\mathsf{a}_R[3,1]\,\mathsf{a}_T[1,2]\,\mathsf{a}_T[1,3]$.

The first interleaving corresponds to an honest execution. First, a tag (with id 1) executes its action (for session id 2). Then, the reader (session id 3) executes its first action for $i = 1$, i.e. it recognizes a valid input w.r.t. $\mathsf{key}[1]$ of the tag with id 1.

The second interleaving does not correspond to any real execution since the conditions of actions $\mathsf{a}_R[3,1]$ and $\mathsf{a}_{R1}[3]$ could not be satisfied simulatenously.

The third interleaving is also not executable: it represents an execution in which the reader (session id 3) recognizes a valid input from tag 1 before any output from tag 1.

As mentioned before, our tool takes as input a protocol specification expressed in a fragment of the applied pi-calculus, and automatically translates it to a protocol according to Definition 3. The translation is rather straightforward but its description and the study of its semantic properties are outside the scope of this paper.

IV. Semantics

The semantics of our meta-logic is given through a translation from the meta-logic to the base logic of [11]. We recall the semantics of the base logic in Section IV-A before defining the translation in Section IV-B.

A. Base Logic

We briefly recall the key definitions of [11], considering only the sort message, and a single attacker symbol att. We do not rely on the way [11] encodes protocol equivalence into the logic (using a so-called folding operation) but only need the core logic for reasoning about computational indistinguishability of sequences of messages.

Syntax: The base logic is a first-order logic, in which terms represent probabilistic PTIME Turing machines producing bitstrings, and a single predicate $\sim$ represents computational indistinguishability. A key idea of the CCSA approach is to use a special attacker function symbol att to represent the attacker's computations, which is left unspecified to model the fact that the attacker may perform any arbitrary probabilistic PTIME computation. The logic is parameterized by a set $\mathcal{N}_B$ of name symbols, a set of variables $\mathcal{X}_B$, and a set of function symbols $\mathcal{F}_B$. Terms are generated from $\mathcal{X}_B$ and $\mathcal{N}_B$ using the unary function symbol att and the function symbols of $\mathcal{F}_B$. We assume that $\mathcal{F}_B$ contains at least the following symbols, with the expected arities and usual notations:

- pairing $\langle \_, \_ \rangle$, equality $\mathsf{EQ}(\_, \_)$;
- constants empty, true and false;
- conditionals $\mathsf{if}\ \_\ \mathsf{then}\ \_\ \mathsf{else}\ \_$.

We do not use a predicate symbol for equality in the base logic: $\mathsf{EQ}(u, v)$ is a term and we may write, for instance, $\mathsf{EQ}(\mathsf{true}, \mathsf{EQ}(u, v))$. We allow ourselves to use the same notations for some constructs in the meta-logic and base logic, because our translation is homomorphic w.r.t. them.

Atomic formulas are of the form $u_1, \ldots, u_n \sim v_1, \ldots, v_n$ where $n \geq 0$ and $u_1, \ldots, u_n$, $v_1, \ldots, v_n$ are terms, and represent indistinguishabilities between two experiments.

Semantics: We are interested in the interpretation of formulas of the base logic in a specific class of first-order interpretations, called *computational models*. The domain of a computational model $\mathbb{M}$ is the set of PTIME Turing machines receiving as inputs the security parameter η in unary (1^η) and a pair $\rho = (\rho_s, \rho_r)$ of random tapes (the complexity is w.r.t. the security parameter, not the random tapes). The tape ρ_s is used to draw honestly generated random values, and is not directly accessible by the attacker, and ρ_r is used for random values drawn by the attacker. The interpretation $[\![t]\!]$ of a term as a Turing machine is defined as follows.

- Each name $\mathsf{n} \in \mathcal{N}_B$ is interpreted as a machine that extracts a word of length η from ρ_s, such that different names extract disjoint parts of the tape.
- The symbols empty, true, false, EQ and $\mathsf{if}\ \_\ \mathsf{then}\ \_\ \mathsf{else}\ \_$ are interpreted in the expected way. E.g., for any terms t_1, t_2, $[\![\mathsf{EQ}(t_1, t_2)]\!]$ is the Turing machine that, on input $(1^\eta, \rho)$, returns 1 if $[\![t_1]\!]$ and $[\![t_2]\!]$ return the same result:
$$[\![\mathsf{EQ}(t_1, t_2)]\!](1^\eta, \rho) = \begin{cases} 1 & \text{if } [\![t_1]\!](1^\eta, \rho) = [\![t_2]\!](1^\eta, \rho) \\ 0 & \text{otherwise} \end{cases}$$
- The other function symbols in $\mathcal{F}_B$ are interpreted as arbitrary PTIME Turing machines that do not access the random tapes. When studying a specific protocol, we restrict computational models according to the assumptions the protocol relies on: e.g. we may assume that a binary function symbol $\oplus$ is interpreted as exclusive or, that a binary function symbol H is interpreted as a PRF keyed hash function, ...
- The symbol att is interpreted as a PTIME Turing Machine that does not access the random tape ρ_s, but has access to ρ_r.

Finally, the predicate $\sim$ is interpreted as computational indistinguishability (noted $\approx$), where $d_1, \ldots, d_n \approx d'_1, \ldots, d'_n$ when for any PTIME Turing machine $\mathcal{A}$,

$$\begin{aligned} | \, &\mathbf{Pr}(\rho : \mathcal{A}(d_1(1^\eta, \rho), \ldots, d_n(1^\eta, \rho), \rho_r) = 1) - \\ &\mathbf{Pr}(\rho : \mathcal{A}(d'_1(1^\eta, \rho), \ldots, d'_n(1^\eta, \rho), \rho_r) = 1) \, | \end{aligned}$$

is negligible in η.

We write $\mathbb{M} \models \phi$ when the base formula ϕ is satisfied in the computational model $\mathbb{M}$, and we say that ϕ is valid if it is satisfied in any computational model.

Example 8. *Assume that* n *and* m *are two distinct names. The formulas* $\mathsf{n} \sim \mathsf{m}$ *and* $\mathsf{EQ}(\mathsf{n}, \mathsf{m}) \sim \mathsf{false}$ *are valid: indeed, the attacker cannot distinguish between two random samplings with the same distribution, and there is a negligible probability that two independent uniform samplings of length* η *coincide.*

The if _ then _ else _ function symbol allows to define other boolean constructs. We write $u \mathbin{\dot\wedge} v$ for $\mathsf{if}\ u\ \mathsf{then}\ v\ \mathsf{else}\ \mathsf{false}$, and define similarly $u \mathbin{\dot\vee} v$ and $u \mathbin{\dot\Rightarrow} v$. Finally, we write $u \doteq v$ for $\mathsf{EQ}(u, v)$.

Example 9. *Consider the following base logic formulas:*

$$(u \sim \mathsf{true}) \Rightarrow (v \sim \mathsf{true}) \quad (a)$$

$$(u \mathbin{\dot\Rightarrow} v) \sim \mathsf{true} \qquad (b)$$

Formula (a) is a logical consequence of (b): if both $u \mathbin{\dot\Rightarrow} v$ *and* u *are true with overwhelming probability, then it must also be the case for* v.

However, (a) does not generally imply (b). Consider a unary function symbol f *and a model* $\mathbb{M}$ *where* f *is interpreted as the machine that returns the first bit of its argument. Then, for any arbitrary name* n*, the term* $f(\mathsf{n})$ *is interpreted as the probabilistic computation returning* 1 *with probability* $\frac{1}{2}$*, and* 0 *otherwise. We have* $\mathbb{M} \not\models (f(\mathsf{n}) \sim \mathsf{true})$ *hence formula (a) is satisfied in* $\mathbb{M}$ *when* $u := f(\mathsf{n})$*, regardless of* v*. However,* $\mathbb{M} \not\models (f(\mathsf{n}) \mathbin{\dot\Rightarrow} \mathsf{false}) \sim \mathsf{true}$*. In other words,* $f(\mathsf{n})$ *is not true with overwhelming probability, but it is also not false with overwhelming probability.*

B. Translation

Our translation from the meta-logic to the base logic, is parameterized by the protocol that the meta-logic is meant to describe. From now on, we assume some protocol $\mathcal{P} = (\mathcal{P}_{\mathcal{A}}, <)$ built upon a set of actions $\mathcal{A}$ using function and name symbols from $\mathcal{F}$ and $\mathcal{N}$. Therefore, we consider meta-logic terms and formulas over $\Sigma = (\mathcal{F}, \mathcal{N}, \mathcal{A})$.

Definition 6. *Given a finite set* D *of integers, the base logic signature* $\Sigma^D = (\mathcal{F}_B, \mathcal{N}_B)$ *contains exactly:*

- *a name symbol* $\mathsf{n}_{k_1,\dots,k_p}$ *for every* $\mathsf{n} \in \mathcal{N}$ *of index arity* p*, and every* $k_1, \dots, k_p \in D$*;*
- *a function symbol* $\mathsf{f}_{k_1,\dots,k_p}$ *of arity* n *for every* $\mathsf{f} \in \mathcal{F}$ *of index arity* p *and message arity* n*, and every* $k_1, \dots, k_p \in D$*.*

Example 10. *In the Basic Hash protocol,* $\mathsf{n} \in \mathcal{N}$ *is a name symbol of our meta-logic, of index arity* 2*. Let* $D = \{1, 2\}$*. We have* $\Sigma^D = (\mathcal{F}_B, \mathcal{N}_B)$ *with* $\mathcal{N}_B = \{\mathsf{n}_{1,1}, \mathsf{n}_{1,2}, \mathsf{n}_{2,1}, \mathsf{n}_{2,2}\}$*. In other words, for this choice of concrete indices, we consider four different names in the base logic. Function symbols used to model primitives are all of index arity* 0*, thus we have a one to one correspondence between function symbols in* $\mathcal{F}$ *and those in* $\mathcal{F}_B$*, and we still write* H*,* $\langle \_, \_ \rangle$*,* fst*, and* snd*.*

We now define the structure that allows us to interpret meta-terms and meta-formulas. The idea is that for each possible interleaving of the protocol, we can define a structure such that the macros at each timestamp correspond to their expected value for that interleaving.

Definition 7. *A trace model* $\mathbb{T}$ *(associated to a protocol* $\mathcal{P}$*) is a tuple* $(\mathcal{D}_{\mathcal{I}}, \mathcal{D}_{\mathcal{T}}, <_{\mathcal{T}}, \sigma_{\mathcal{I}}, \sigma_{\mathcal{T}})$ *such that:*

- $\mathcal{D}_{\mathcal{I}} \subseteq \mathbb{N}$ *is a finite index domain;*
- $<_{\mathcal{T}}$ *is a total ordering on* $\mathcal{D}_{\mathcal{T}} := \{\mathsf{init}\} \uplus \{a[k_1, \dots, k_n] \mid a \in \mathcal{A}, k_1, \dots, k_n \in \mathcal{D}_{\mathcal{I}}\}$ *such that* init *is minimal, and such that the sequence of elements of* $\mathcal{D}_{\mathcal{T}}$ *ordered by* $<_{\mathcal{T}}$ *is an interleaving of* $\mathcal{P}$*;*
- $\sigma_{\mathcal{I}} : \mathcal{I} \to D_{\mathcal{I}}$ *and* $\sigma_{\mathcal{T}} : \mathcal{T} \to D_{\mathcal{T}}$ *are mappings that interpret index and timestamp variables as elements of their respective domains.*

We include $\sigma_{\mathcal{I}}$ and $\sigma_{\mathcal{T}}$ to ease the presentation, which means that trace models provide an interpretation for all index and timestamp variables. This information is often irrelevant: when interpreting a formula, only the interpretation of its free variables will matter.

The total ordering $<_{\mathcal{T}}$ yields a predecessor function $\mathsf{pred}_{\mathcal{T}} : \mathcal{D}_{\mathcal{T}} \to \mathcal{D}_{\mathcal{T}}$ which maps init to itself and all other elements $v \in \mathcal{D}_{\mathcal{T}}$ to the largest $v' \in \mathcal{D}_{\mathcal{T}}$ such that $v' < v$.

Example 11. *Continuing our running example, we consider* $\mathcal{D}_I = \{1\}$*, and the following total ordering:*

$$\mathsf{init} < \mathsf{a}_T[1,1] < \mathsf{a}_R[1,1] < \mathsf{a}_{R1}[1]$$

Therefore, we have that $\mathsf{pred}_{\mathcal{T}}(\mathsf{a}_R[1,1]) = \mathsf{a}_T[1,1]$*. The notion of trace model forces us to include both* $\mathsf{a}_{R1}[1]$ *and* $\mathsf{a}_R[1,1]$ *even though their conditions are mutually exclusive. It is not a concern: what matters is that all real executions of the protocol are accounted for by some prefix of the complete interleaving induced by some trace model.*

When $\mathbb{T} = (\mathcal{D}_{\mathcal{I}}, \mathcal{D}_{\mathcal{T}}, <_{\mathcal{T}}, \sigma_{\mathcal{I}}, \sigma_{\mathcal{T}})$ is a trace model and $k \in \mathcal{D}_{\mathcal{I}}$, $\mathbb{T}\{i \mapsto k\}$ is the trace model identical to $\mathbb{T}$ in which $\sigma_{\mathcal{I}}$ is updated to map i to k. We similarly define $\mathbb{T}\{\tau \mapsto v\}$ when $v \in \mathcal{D}_{\mathcal{T}}$.

We can now define, for each meta-term t and trace model $\mathbb{T}$, the base logic term $(t)^{\mathbb{T}}$, and similarly for formulas. The complete definition is given in Appendix A, and we only present here its general principle. It is defined inductively on the structure of meta-terms and meta-formulas, translating each meta-logic construct by its counterpart in the base logic when it is available. Indexed function symbols are translated to their counterpart in $\Sigma^{\mathcal{D}_{\mathcal{I}}}$. For instance, names are translated as follows:

$$(\mathsf{n}[i_1, \dots, i_p])^{\mathbb{T}} \overset{\text{def}}{=} \mathsf{n}_{\sigma_{\mathcal{I}}(i_1), \dots, \sigma_{\mathcal{I}}(i_p)}.$$

Boolean constructs are translated to their dotted counterparts. Finally, lookup constructs are translated to nested conditionals:

$$\begin{aligned} &(\mathsf{find}\ \vec{i}\ \mathsf{suchthat}\ \phi\ \mathsf{in}\ t\ \mathsf{else}\ t')^{\mathbb{T}} \overset{\text{def}}{=} \\ &\quad \mathsf{if}\ (\phi)^{\mathbb{T}\{\vec{i} \mapsto \vec{k}_1\}}\ \mathsf{then}\ (t)^{\mathbb{T}\{\vec{i} \mapsto \vec{k}_1\}}\ \mathsf{else} \\ &\quad \mathsf{if}\ (\phi)^{\mathbb{T}\{\vec{i} \mapsto \vec{k}_2\}}\ \mathsf{then}\ (t)^{\mathbb{T}\{\vec{i} \mapsto \vec{k}_2\}}\ \mathsf{else} \\ &\quad \dots \\ &\quad \mathsf{if}\ (\phi)^{\mathbb{T}\{\vec{i} \mapsto \vec{k}_p\}}\ \mathsf{then}\ (t)^{\mathbb{T}\{\vec{i} \mapsto \vec{k}_p\}}\ \mathsf{else}\ (t')^{\mathbb{T}} \end{aligned}$$

$$
\begin{aligned}
\mathsf{cond}_{\mathsf{init}} &= \mathsf{exec}_{\mathsf{init}} = \mathsf{true}\\
\mathsf{input}_{\mathsf{init}} &= \mathsf{frame}_{\mathsf{init}} = \mathsf{output}_{\mathsf{init}} = \mathsf{empty}\\
\mathsf{output}_{\mathsf{a}[\vec{i}]} &= o\\
\mathsf{cond}_{\mathsf{a}[\vec{i}]} &= \phi\\
\mathsf{exec}_{\mathsf{a}[\vec{i}]} &= \mathsf{cond}@\mathsf{a}[\vec{i}] \wedge \mathsf{exec}@\mathsf{pred}(\mathsf{a}[\vec{i}])\\
\mathsf{frame}_{\mathsf{a}[\vec{i}]} &= \langle \mathsf{exec}@\mathsf{a}[\vec{i}],\\
&\qquad \langle \mathsf{if}\ \mathsf{exec}@\mathsf{a}[\vec{i}]\ \mathsf{then}\ \mathsf{output}@\mathsf{a}[\vec{i}]\ \mathsf{else}\ \mathsf{empty},\\
&\qquad\quad \mathsf{frame}@\mathsf{pred}(\mathsf{a}[\vec{i}])\rangle\rangle\\
\mathsf{input}_{\mathsf{a}[\vec{i}]} &= \mathsf{att}\big(\mathsf{frame}@\mathsf{pred}(\mathsf{a}[\vec{i}])\big)
\end{aligned}
$$

Fig. 2. Interpretation of macros, where $\mathsf{a}[\vec{i}].(\phi, o)$ is an action of $\mathcal{P}_\mathcal{A}$.

where $\vec{k}_1, \ldots, \vec{k}_p$ is a complete enumeration of $\mathcal{D}_\mathcal{I}^{|\vec{i}|}$.

Regarding meta-formulas, quantifications over index and timestamp variables do not have a direct counterpart in the base logic. They are translated to finite boolean expressions through nested conditionals relying on $\dot\wedge$ and $\dot\vee$ introduced previously:

$$
\begin{aligned}
(\forall i.\phi)^\mathbb{T} &\overset{\text{def}}{=} \dot{\bigwedge}_{k\in D_\mathcal{I}} (\phi)^{\mathbb{T}\{i\mapsto k\}}\\
(\forall \tau.\phi)^\mathbb{T} &\overset{\text{def}}{=} \dot{\bigwedge}_{v\in D_\mathcal{T}} (\phi)^{\mathbb{T}\{\tau\mapsto v\}}
\end{aligned}
$$

and similarly for existential quantifications.

Finally, we also have to give a meaning to the macros input, output, frame, cond, and exec used in the meta-logic. We define in Fig. 2 the terms m_{init} and $\{m_{\mathsf{a}[\vec{i}]} \mid \mathsf{a}[\vec{i}] \in \mathcal{P}_\mathcal{A}\}$ for each of these macro symbols, and then have:

$$
(m@T)^\mathbb{T} \overset{\text{def}}{=} \begin{cases} (m_{\mathsf{init}})^\mathbb{T} & \text{if } (T)^\mathbb{T} = \mathsf{init}\\ (m_{\mathsf{a}[\vec{i}]})^{\mathbb{T}\{\vec{i}\mapsto\vec{k}\}} & \text{if } (T)^\mathbb{T} = \mathsf{a}[\vec{k}] \text{ and } \mathsf{a}[\vec{i}] \in \mathcal{P}_\mathcal{A} \end{cases}
$$

Roughly, an output macro is replaced by the meta-term as specified by the protocol and it is then interpreted in the trace model to get a base term. The cond macro has a similar treatment and produces a base formula corresponding to the conditional of the action. The exec macro simply corresponds to the conjunction of all past conditions.

The translation of the frame gathers (using nested pairs) all the information available to the attacker at some execution point: for each past action, the attacker observes if the execution continues and, if that is the case, they obtain the output. Finally, in order to model the attacker's capabilities, the input macro is interpreted using the attacker symbol att, to which we pass the current frame.

Example 12. *Considering the trace model $\mathbb{T}$ given in Example 11 with $\sigma_\mathcal{I}(i) = \sigma_\mathcal{I}(j) = 1$, we give below the interpretation of several meta-terms:*

$$
\begin{aligned}
&(\mathsf{output}@\mathsf{a}_T[i,j])^\mathbb{T} = \langle \mathsf{n}_{1,1}, \mathsf{H}(\mathsf{n}_{1,1}, \mathsf{key}_1)\rangle \overset{\textit{def}}{=} t_{\mathsf{out}}\\
&(\mathsf{fst}(\mathsf{output}@\mathsf{a}_T[i,j]))^\mathbb{T} = \mathsf{fst}(t_{\mathsf{out}})\\
&(\mathsf{input}@\mathsf{a}_R[j,i])^\mathbb{T} =\\
&\quad \mathsf{att}(\langle \mathsf{true}, \langle \mathsf{if}\ \mathsf{true}\ \mathsf{then}\ t_{\mathsf{out}}\ \mathsf{else}\ \mathsf{empty}, \mathsf{empty}\rangle\rangle) \overset{\textit{def}}{=} t_{\mathsf{in}}
\end{aligned}
$$

Therefore the meta-formula expressing an authentication property in Listing 2 translates as follows in $\mathbb{T}$:

$$
\begin{aligned}
&\mathsf{snd}(t_{\mathsf{in}}) \doteq \mathsf{H}(\mathsf{fst}(t_{\mathsf{in}}), \mathsf{key}_1) \;\; \dot\Rightarrow\\
&\qquad \big(\mathsf{true}\ \dot\wedge\ \mathsf{fst}(t_{\mathsf{in}}) \doteq \mathsf{fst}(t_{\mathsf{out}})\ \dot\wedge\ \mathsf{snd}(t_{\mathsf{in}}) \doteq \mathsf{snd}(t_{\mathsf{out}})\big)
\end{aligned}
$$

Note that the atom $\mathsf{a}_T[i,j'] \leq \mathsf{a}_R[j,i]$ has been replaced by true *because it holds in $\mathbb{T}$ when i, j and j' are (necessarily) interpreted as 1. If we had had $\mathsf{a}_T[1,1] > \mathsf{a}_R[1,1]$ in $\mathbb{T}$ we would have obtained* false*, making the conclusion of the implication unsatisfiable.*

The base formula of the previous example corresponds to the translation of the meta-formula expressing authentication considering a single and very simple trace model. To provide a proof of our authentication property, we have to verify the validity of the base formula obtained with any trace model.

C. Validity

We say that a meta-formula ϕ is satisfied in $\mathbb{T}, \mathbb{M}$, written $\mathbb{T}, \mathbb{M} \models_\mathcal{P} \phi$, or simply $\mathbb{T}, \mathbb{M} \models \phi$ when $\mathcal{P}$ is clear from the context, whenever $\mathbb{M} \models (\phi)^\mathbb{T} \sim \mathsf{true}$. Intuitively, it means that the formula ϕ is true with overwhelming probability on the trace $\mathbb{T}$. Then, we say that ϕ is valid if it is satisfied in every $\mathbb{T}$ and $\mathbb{M}$: intuitively, the formula ϕ is true with overwhelming probability for every executions of the associated protocol $\mathcal{P}$.

In practice, this notion of validity is too strong: for instance, the authentication property of the Basic Hash protocol only holds if the hash satisfies some unforgeability assumption. Thus, we are interested in verifying the validity of meta-formulas for restricted classes $\mathcal{C}$ of models. We will consider two types of restrictions. First, we may make some security assumptions on the interpretation of cryptographic primitives in $\mathbb{M}$: e.g., when a hash function is declared in our prover, we assume that its interpretation satisfies the PRF assumption. Second, further assumptions can be made by adding axioms expressed as meta-formulas. An axiom ϕ will restrict the considered class $\mathcal{C}$ to those $\mathbb{T}$ and $\mathbb{M}$ such that $\mathbb{M}, \sigma \models (\phi)^\mathbb{T} \sim \mathsf{true}$ for all semantic assignments σ mapping free message variables occurring in ϕ to probabilistic PTIME machines. Axioms are used, for example, to express properties of message lengths: $\mathsf{len}(\langle x, y\rangle) = \mathsf{plus}(\mathsf{len}(x), \mathsf{len}(y))$.

Note that our notion of validity differs from standard notion of security in the computational model. In our logic, if a formula ϕ is valid, it means that for any given trace and for any attacker interacting with the protocol along this trace, ϕ is false with negligible probability. In the computational model, one would rather expect that for any attacker interacting with the protocol and choosing a polynomial number of actions to execute, ϕ is false with negligible probability. In the former case, the advantage of the attacker may grow super-polynomially w.r.t. to the number of sessions, but not in the latter. We provide a contrived example showing the difference between the two notions in Appendix D. While weaker than concrete security bounds, we stress that this guarantee is stronger than symbolic guarantees for unbounded sessions. Furthermore, this limitation can be lifted using the composition result of [27], as done in some of our case studies (cf. Section VII).

$$\textsc{NameIndep}\quad \frac{\mathsf{n} \neq \mathsf{m}}{\Gamma, \mathsf{n}[\vec{i}] = \mathsf{m}[\vec{j}] \vdash \phi} \qquad \textsc{NameEq}\quad \frac{\Gamma, i_1 = j_1, \ldots, i_k = j_k \vdash \phi}{\Gamma, \mathsf{n}[i_1, \ldots, i_k] = \mathsf{n}[j_1, \ldots, j_k] \vdash \phi}$$

$$\textsc{ActDep}\quad \frac{\textit{when } \mathsf{a}[\vec{j}] < \mathsf{b}[\vec{i}]}{\Gamma, \mathsf{b}[\vec{i}] \leq \mathsf{a}[\vec{j}] \vdash \phi} \qquad \textsc{ActIndep}\quad \frac{\textit{when } \mathsf{a} \neq \mathsf{b}}{\Gamma, \mathsf{a}[\vec{i}] = \mathsf{b}[\vec{j}] \vdash \phi}$$

$$\textsc{ActEq}\quad \frac{\Gamma, i_1 = j_1, \ldots, i_k = j_k \vdash \phi}{\Gamma, \mathsf{a}[i_1, \ldots, i_k] = \mathsf{b}[j_1, \ldots, j_k] \vdash \phi} \qquad \textsc{Exec}\quad \frac{\Gamma, \forall \tau' \leq \tau.\mathsf{cond}@\tau' \vdash \phi}{\Gamma, \mathsf{exec}@\tau \vdash \phi}$$

$$\textsc{Init}\quad \frac{}{\Gamma, \tau \neq \mathsf{init} \wedge \tau \leq \mathsf{pred}(\tau) \vdash \phi} \qquad \textsc{Pred}\quad \frac{\Gamma, \tau' = \mathsf{pred}(\tau) \vee \tau' = \tau \vdash \phi}{\Gamma, \mathsf{pred}(\tau) \leq \tau' \leq \tau \vdash \phi}$$

Fig. 3. Some rules of our sequent calculus for reachability.

Meta-formulas express properties of all execution traces of a protocol. Some security properties (e.g. strong secrecy, unlinkability) are better expressed as equivalences between two protocols. We accomodate such notions naturally in our framework since it is based on an indistinguishability predicate: this is presented in Section VI. Before that, we design in the next section a proof system that allows to derive valid meta-formulas (relatively to some classes of models).

V. Reachability Rules

We now present our reachability sequent calculus, for some protocol $\mathcal{P} = (\mathcal{P}_{\mathcal{A}}, <)$ fixed throughout the section.

Definition 8. *A sequent $\Gamma \vdash \phi$ is formed from a set of meta-formulas Γ and a meta-formula ϕ, both without message variables. The sequent $\Gamma \vdash \phi$ is valid w.r.t. a class of models $\mathcal{C}$ if $(\bigwedge \Gamma) \Rightarrow \phi$ is valid w.r.t. $\mathcal{C}$.*

Definition 9. *An inference rule $\dfrac{\Gamma_1 \vdash \phi_1 \ \ldots \ \Gamma_n \vdash \phi_n}{\Gamma \vdash \phi}$ is sound w.r.t. a class $\mathcal{C}$ when the conclusion is valid w.r.t. $\mathcal{C}$ whenever the premises are valid w.r.t. $\mathcal{C}$.*

We now give our meta-logic reachability sequent calculus rules, and prove their soundness. We talk of validity (resp. soundness) without specifying the class $\mathcal{C}$ when it holds w.r.t. all models.

A. Basic Rules

Although our sequents have a probabilistic semantics, all rules of *classical* first-order sequent calculus are sound.

We give in Fig. 3 the rules we designed which are specific to security protocol analysis, and describe them below. Two different names are almost never equal (i.e. they are not equal except for a negligible number of samplings). This can be the case either because they have different head symbols (rule NameIndep) or because they have the same head symbols but different indices (rule NameEq). The rule ActDep states that actions must occur in the order imposed by the protocol, and ActIndep and ActEq express that two different actions cannot occur at the same instant. The Pred rule states that there is no timestamp between $\mathsf{pred}(\tau)$ and τ, and Exec states that if the trace is executable up to τ then the protocol conditions hold for any instant before τ. Init says that if τ is not the initial timestamp, then τ cannot occur before the action preceding it.

$$\oplus\textsc{-Nil}\quad \frac{}{\Gamma \vdash t \oplus t = 0} \qquad \oplus\textsc{-Sym}\quad \frac{}{\Gamma \vdash t \oplus t' = t' \oplus t}$$

$$\oplus\textsc{-Assoc}\quad \frac{}{\Gamma \vdash t \oplus (t' \oplus t'') = (t \oplus t') \oplus t''}$$

Fig. 4. Some XOR rules of our sequent calculus for reachability.

We also have rules for expanding macros into their meaning. For instance, if $\mathcal{P}_{\mathcal{A}}$ contains the action $\mathsf{a}[\vec{i}].(\phi, o)$, we can derive $\Gamma \vdash \mathsf{output}@\mathsf{a}[\vec{i}] = o$ and $\Gamma \vdash \mathsf{cond}@\mathsf{a}[\vec{i}] \Leftrightarrow \phi$. All these rules are sound.

Some of our rules are sound only under some computational assumptions. These assumptions can either be cryptographic assumptions (we give examples in the next section), or functional properties of the primitives. E.g, the rules $\oplus$-Nil, $\oplus$-Sym and $\oplus$-Assoc of Fig. 4 state functional properties that $\oplus$ satisfies whenever it is interpreted as the XOR operator: these rules are obviously sound in any computational model where $\oplus$ is interpreted as XOR.

B. Advanced Rules

We now describe how we designed the most advanced rules of our reachability sequent calculus, which deal with cryptographic assumptions and probabilistic independence. We present here the rule for the EUF-CMA axiom.

a) Base logic rule: We recall the base logic EUF-CMA rule from [41]. Before starting, we introduce notations we use to describe syntactic side-conditions of rules.

Definition 10. *A template $C[[]_1, \ldots, []_n, \bullet]$ is a syntactic expression built using the hole variables $([]_i)_{1 \leq i \leq n}$, the special variable $\bullet$, and applications of function symbols $\mathsf{f} \in \mathcal{F}_B$ (with the correct arity) such that $\bullet$ occurs* exactly once *in C.*

For any base terms $t_1, \ldots, t_n, t$, we let $C[t_1, \ldots, t_n, t]$ be the base term obtained by substituting the hole variables and $\bullet$ by the terms $t_1, \ldots, t_n, t$.

When possible, we omit the hole variables and write $C[\_, \bullet]$.

Definition 11. *Let $C[\_, \bullet]$ be a template, u a* ground *base term and n a name. Then $\mathsf{n} \sqsubseteq_C u$ holds whenever n appears in u only in subterms of the form $C[\vec{w}, \mathsf{n}]$ for some $\vec{w}$.*

Example 13. *We give two examples:*

- $\mathsf{n} \not\sqsubseteq_{\bullet} u$ *states that* n *does not appear in* u*.*
- $\mathsf{k} \sqsubseteq_{\mathsf{H}(\_,\bullet)} u$ *states that* k *appears only as a hash key in* u*.*

Roughly, the EUF-CMA rule states that if s is a valid hash of m, then m must be equal to some honestly hashed message appearing in s or m. We formally define it next, as a simple elaboration of the rule EUF-MAC of [41] (a base logic sequent $\Gamma \vdash \phi$ is valid when $((\dot{\bigwedge} \Gamma) \Rightarrow \phi) \sim \mathsf{true}$ is valid).

Definition 12. *For any* ground *base terms s, u, m and name k, we let* EUF-CMA *be the rule:*

$$\frac{\Gamma, \dot{\bigvee}_{\mathsf{H}(u,\mathsf{k}) \in \mathsf{st}(s,m)} m \doteq u \vdash \phi}{\Gamma, s \doteq \mathsf{H}(m, \mathsf{k}) \vdash \phi} \quad \textit{when } \mathsf{k} \sqsubseteq_{\mathsf{H}(\_,\bullet)} s, m$$

Example 14. *If u is the term $\langle \mathsf{H}(t_1, \mathsf{k}), \langle \mathsf{H}(t_2, \mathsf{k}), \mathsf{H}(t_3, \mathsf{k}')\rangle\rangle$ where k, k' are names and s, t_0, t_1 and t_2 are ground terms that do not use k and k', then we have an* EUF-CMA *instance:*

$$\frac{\Gamma, (\mathsf{att}(u) \doteq t_1) \mathbin{\dot{\vee}} (\mathsf{att}(u) \doteq t_2) \vdash \phi}{\Gamma, s \doteq \mathsf{H}(\mathsf{att}(u), \mathsf{k}) \vdash \phi}$$

Indeed, k appears only in hash positions in s and $\mathsf{att}(u)$, hence $\mathsf{k} \sqsubseteq_{\mathsf{H}(\_,\bullet)} s, \mathsf{att}(u)$. Moreover, t_1 and t_2 are the only messages hashed by k (note that t_3 is hashed by a different key k').

b) Meta-logic rule: We now explain how to lift the base logic rule EUF-CMA to the meta-logic. We need to find a rule such that its translation in any trace model yields a valid instance of the base logic EUF-CMA rule. There are two obstacles to this. Assume our meta-logic rule is of the following form:

$$\frac{\Gamma, \dot{\bigvee}_{\mathsf{H}(u,\mathsf{k}[\vec{i}]) \in \mathsf{ST}(s,m)} m \doteq u \vdash \phi}{\Gamma, s \doteq \mathsf{H}(m, \mathsf{k}[\vec{i}]) \vdash \phi} \text{ when } \mathsf{SSC}_{\mathsf{k}[\vec{i}]}(s, m) \qquad (1)$$

Soundness requires that:

- if the *meta-logic* side-condition $\mathsf{SSC}_{\mathsf{k}[\vec{i}]}(s, m)$ holds, then all translations of s and m must satisfy the *base logic* side-condition:
$$\forall \mathbb{T}, (\mathsf{k}[\vec{i}])^{\mathbb{T}} \sqsubseteq_{\mathsf{H}(\_,\bullet)} (s)^{\mathbb{T}}, (m)^{\mathbb{T}} \qquad (2)$$
- the set of meta-terms $\mathsf{ST}(s, m)$ must be such that its translation in any trace model contains all hashes of the translation of s and m. That is, for every $\mathbb{T}$:
$$\left\{\mathsf{H}(u, (\mathsf{k}[\vec{i}])^{\mathbb{T}}) \in \mathsf{st}((s)^{\mathbb{T}}, (m)^{\mathbb{T}})\right\} \subseteq (\mathsf{ST}(s, m))^{\mathbb{T}} \qquad (3)$$

Note that both obstacles are of the same nature, though there is an additional difficulty in the second condition: since the set on the left of (3) can be arbitrarily large (because the trace in $\mathbb{T}$ can be arbitrarily long), the set $(\mathsf{ST}(s, m))^{\mathbb{T}}$ may be infinite (because of this, our final rule will slightly differ from the form proposed in (1)).

We start by focusing on the occurrence side condition $\sqsubseteq_{\mathsf{H}(\_,\bullet)}$. First, we naturally lift it to meta-terms, by requiring that the side-condition holds for the translation of the meta-terms in any trace model.

Definition 13. *Let C be a template and u a meta-term with* no message variables. *For any* name symbol n, *we let $\mathsf{n} \sqsubseteq^{\mathcal{P}}_{C} u$ hold whenever $\mathsf{n}_{\vec{k}} \sqsubseteq_{C} (u)^{\mathbb{T}}_{\mathcal{P}}$ holds for any trace model $\mathbb{T}$ and $\vec{k} \subseteq \mathcal{D}_{\mathcal{I}}$ (of length arity of n).*

The base logic side-condition $\sqsubseteq_C$ is fully syntactic, and can easily be implemented. This is no longer the case with $\mathsf{n} \sqsubseteq^{\mathcal{P}}_{C} u$, as it requires to check a property on all translations of u. Therefore, instead of checking directly $\mathsf{n} \sqsubseteq^{\mathcal{P}}_{C} u$, we are going to check the property on the direct occurrences of n in u (i.e. $\mathsf{n} \sqsubseteq_{C} u$), and on the occurrences of n in any action of the protocol. The idea is that this must over-approximate all occurrences of n in any translation of u in a trace model $\mathbb{T}$.

First, we adapt Definition 11 to meta-terms (base logic function symbols in a template C are seen as function symbols of index arity 0 of the meta-logic).

Definition 14. *Let C be a template, u a meta-term without message variables, and n be a name symbol. Then $\mathsf{n} \sqsubseteq_{C} u$ holds whenever n appears in u only in subterms of the form $C[\vec{w}, \mathsf{n}[\vec{i}]]$ for some $\vec{w}$ and indices $\vec{i}$.*

Definition 15. *Let C be a template, and n be a name symbol. We write $\mathsf{n} \sqsubseteq_{C} \mathcal{P}$ when $\mathsf{n} \sqsubseteq_{C} \{\phi, o \mid \mathsf{a}[\vec{i}].(\phi, o) \in \mathcal{P}_{\mathcal{A}}\}$.*

We give a sufficient condition to check that $\mathsf{n} \sqsubseteq^{\mathcal{P}}_{C} u$.

Proposition 1. *Let C be a template. For any name symbol n and meta-term u, if $\mathsf{n} \sqsubseteq_{C} u$ and $\mathsf{n} \sqsubseteq_{C} \mathcal{P}$ then $\mathsf{n} \sqsubseteq^{\mathcal{P}}_{C} u$.*

Next, we need to over-approximate the (possibly infinite) set of all honest hashes that can appear in the translation of a meta-term, as stated in (3). As for $\sqsubseteq^{\mathcal{P}}_{C}$, we look for all possible occurrences of $\mathsf{H}(u, \mathsf{k})$ either directly in (s, m), or in an action of $\mathcal{P}$. In the latter case, we also construct a formula of the logic that characterizes the fact that the action must have happened before the moment where u was computed, which improves the precision. We capture this through some set $\bar{\mathsf{st}}_{\mathcal{P}}(s, m)$, which is formally defined in [6]. Essentially, $\bar{\mathsf{st}}_{\mathcal{P}}(s, m)$ is a set of triples $(u, \vec{i}, c)$ such that, for any $\mathbb{T}$, any subterm of $(s, m)^{\mathbb{T}}$ that is a hash is the interpretation of some $(u)^{\mathbb{T}\{\vec{i} \mapsto \vec{k}\}}$ such that c holds in $\mathbb{T}\{\vec{i} \mapsto \vec{k}\}$. Moreover, $\vec{i}$ are the new indices in u and c: $\mathsf{fv}(u, c) \subseteq \mathsf{fv}(s, m) \cup \{\vec{i}\}$.

We can now state the meta-logic version of the EUF-CMA rule given in Definition 12.

Proposition 2. *The following rule is sound whenever H is interpreted as an* EUF-CMA *keyed hash-function:*

$$\frac{\Gamma, \bigvee_{(\mathsf{H}(u,\mathsf{k}[\vec{j_0}]),\vec{i},c) \in \bar{\mathsf{st}}_{\mathcal{P}}(s,m)} \exists \vec{i}.(\vec{j_0} = \vec{j} \wedge c \wedge m = u) \vdash \phi}{\Gamma, s = \mathsf{H}(m, \mathsf{k}[\vec{j}]) \vdash \phi}$$

when $\mathsf{k} \sqsubseteq_{\mathsf{H}(\_,\bullet)} \mathcal{P}$ and $\mathsf{k} \sqsubseteq_{\mathsf{H}(\_,\bullet)} s, m$.

Proof (sketch). It suffices to show that, for any trace model $\mathbb{T}$, the translation of the rule in $\mathbb{T}$ is (up to some minor details) an instance of the base logic EUF-CMA rule.

Using Proposition 1, we know that $\mathsf{k} \sqsubseteq^{\mathcal{P}}_{\mathsf{H}(\_,\bullet)} t, m$, hence $\mathsf{k}_{\vec{k}} \sqsubseteq_{\mathsf{H}(\_,\bullet)} (t, m)^{\mathbb{T}}$ for any $\vec{k} \subseteq \mathcal{D}_{\mathcal{I}}$: this guarantees that the side-condition of the base logic rule holds. Then, we show that the translation in $\mathbb{T}$ of the disjunction in the premise of the meta-rule covers all cases of the premise of the base logic EUF-CMA rule, using the fact that $\bar{\mathsf{st}}_{\mathcal{P}}(s, m)$ correctly over-approximates the hashes in $\mathsf{st}((s, m)^{\mathbb{T}})$. □

Example 15. *We illustrate our rule on an example in the context of the Basic Hash protocol, as described in Example 5* *(we factorize the quantification and the equality atom, which are shared by all three cases):*

$$\frac{\begin{array}{l}\Gamma,\ \exists i_0, j_0.\ i_0 = i \wedge \\ \big(\quad (\mathsf{a}_T[i_0,j_0] < \tau \wedge \mathsf{n}[i,j] = \mathsf{n}[i_0,j_0]) \\ \ \vee (\mathsf{a}_R[j_0,i_0] < \tau \wedge \mathsf{n}[i,j] = \mathsf{fst}(\mathsf{input}@\mathsf{a}_R[j_0,i_0])) \\ \ \vee (\mathsf{a}_{R1}[j_0] \quad < \tau \wedge \mathsf{n}[i,j] = \mathsf{fst}(\mathsf{input}@\mathsf{a}_{R1}[j_0]))\quad \big) \quad \vdash \phi\end{array}}{\Gamma, \mathsf{snd}(\mathsf{input}@\tau) = \mathsf{H}(\mathsf{n}[i,j], \mathsf{key}[i]) \vdash \phi}$$

The rule does express that if a hash of $\mathsf{n}[i,j]$ *has been obtained, this message must have been previously hashed. The three possible hashed messages detected by* $\bar{\mathsf{st}}_{\mathcal{P}}(s, m)$ *are restricted to plausible situations: equalities* $i_0 = i$ *ensure that the hashing key is* $\mathsf{key}[i]$ *and inequalities over timestamps (e.g.* $\mathsf{a}_T[i_0, j_0] < \tau$*) ensure that the situation comes from an action that has been previously executed.*

VI. Equivalence Rules

We now turn to proving equivalences. Ultimately, we are interested in proving observational equivalence between two protocols, as this allows to model several security properties such as strong secrecy, anonymity, or unlinkability. We first define the notion of observational equivalence, then define what are our equivalence sequents, and finally present our equivalence sequent calculus.

Intuitively, two protocols $\mathcal{P}_1$ and $\mathcal{P}_2$ are observationally equivalent when they offer the same set of actions for the attacker to execute and, for any sequence of actions that the attacker may decide to execute, the resulting frames with $\mathcal{P}_1$ and $\mathcal{P}_2$ are indistinguishable. In particular, the actions execute with the same probability on both sides, and the sequences of messages that are outputted are indistinguishable. We start by defining the notion of compatible protocols, capturing the idea that they offer the same set of actions.

Definition 16. *Protocols* $\mathcal{P}_1 = (\mathcal{P}^1_{\mathcal{A}}, <_1)$ *and* $\mathcal{P}_2 = (\mathcal{P}^2_{\mathcal{A}}, <_2)$ *are compatible if they are based on the same set* $\mathcal{A}$ *of action names and have the same partial orders, i.e.*

$$\alpha <_1 \alpha' \textit{ if, and only if, } \alpha <_2 \alpha' \textit{ for all } \alpha \textit{ and } \alpha'.$$

Since the notion of trace model only depends on the underlying set of names and partial order, and not on the actual semantics of the protocol given by $\mathcal{P}_{\mathcal{A}}$, we immediately have that two compatible protocols have the same trace models.

Definition 17. *Two protocols* $\mathcal{P}_1$ *and* $\mathcal{P}_2$ *are observationally equivalent if they are compatible and, for any trace model* $\mathbb{T}$*, the base logic formula* $(\mathsf{frame}@\tau)^{\mathbb{T}}_{\mathcal{P}_1} \sim (\mathsf{frame}@\tau)^{\mathbb{T}}_{\mathcal{P}_2}$ *is valid.*

We need a notion of sequent for equivalences that allows us to express observational equivalences. More generally, we want to reason about indistinguishabilities between sequences of meta-level terms and formulas interpreted w.r.t. different protocols on each side of the equivalence.

Definition 18. *Let* $\mathcal{P}_1$ *and* $\mathcal{P}_2$ *be two compatible protocols. A meta-equivalence is an element of the form* $\vec{u} \sim \vec{v}$ *where* $\vec{u}$ *and* $\vec{v}$ *are sequences of meta-terms and meta-formulas of the same length and without message variables. The interpretation of a meta-equivalence in a trace model w.r.t.* $\mathcal{P}_1$ *and* $\mathcal{P}_2$ *is:*

$$(\vec{u} \sim \vec{v})^{\mathbb{T}}_{\mathcal{P}_1,\mathcal{P}_2} \overset{def}{=} (\vec{u})^{\mathbb{T}}_{\mathcal{P}_1} \sim (\vec{v})^{\mathbb{T}}_{\mathcal{P}_2}.$$

An equivalence sequent for $\mathcal{P}_1$ *and* $\mathcal{P}_2$ *is a judgment of the form* $\Delta \vdash_{\mathcal{P}_1,\mathcal{P}_2} E$ *where* E *is a meta-equivalence and* Δ *is a set of meta-equivalences. We may note* $\Delta \vdash E$ *when* $\mathcal{P}_1, \mathcal{P}_2$ *is clear from the context.*

The sequent $\Delta \vdash_{\mathcal{P}_1,\mathcal{P}_2} E$ *is valid when* $\mathbb{T}, \mathbb{M} \models (E)^{\mathbb{T}}_{\mathcal{P}_1,\mathcal{P}_2}$ *for all* $\mathbb{T}$ *and* $\mathbb{M}$ *such that* $\mathbb{T}, \mathbb{M} \models (E')^{\mathbb{T}}_{\mathcal{P}_1,\mathcal{P}_2}$ *for each* $E' \in \Delta$.

We now present our equivalent calculus, for some compatible protocols $\mathcal{P}_1$ and $\mathcal{P}_2$, fixed for the rest of this section.

A. Basic Rules

We present in Fig. 5 some basic inference rules for deriving equivalences, whose soundness does not depend on any cryptographic assumption. In these rules, we use the meta-variable ξ to denote either a meta-term or a meta-formula.

All the rules but Induction are obvious liftings of known axioms for deriving equivalences in the base logic (see e.g. [11]). E.g., for the Refl rule: if $\vec{u}$ is macro-free then its interpretations w.r.t. $\mathcal{P}_1$ and $\mathcal{P}_2$ are the same, thus our rule is, for each possible trace interpretation, an instance of the reflexivity rule of the base logic. It is also the case of rules Enrich, Dup and FA. Our proof system allows variants of the FA rule, such as FA-$\Diamond$, which are still liftings of the base logic FA rule. Finally, rule Equiv-Term allows to replace some occurrences of a meta-term t by t' on the left side of the equivalence provided that $\vdash_{\mathcal{P}_1} t = t'$ is derivable. We can similarly replace a meta-formula by an equivalent one using Equiv-Form. Obviously, the variants of these rules working on the right side of equivalences are also allowed.

The Induction rule allows to prove an equivalence for any arbitrary timestamp τ by proving the equivalence when τ is init and, for each action $\mathsf{a}[\vec{i}]$ of the protocol, when τ is $\mathsf{a}[\vec{i}]$ assuming that the equivalence holds for $\mathsf{pred}([\vec{i}])$. Proofs of observational equivalence almost always start with this rule.

B. Advanced Rules

We now present some more advanced rules. The soundness arguments for these rules are postponed to Appendix C.

We show in Fig. 6 our rule Fresh. It is based on the base logic rule which states that adding fresh names on each side of an equivalence preserves its validity: indeed, these independent uniform random samplings do not bring any new information to the attacker. We lift this at the meta-level by over-approximating the freshness condition as meta-formulas: we have $(\mathsf{n}[\vec{i}])^{\mathbb{T}}_{\mathcal{P}_1} \not\in \mathsf{st}((\vec{u})^{\mathbb{T}}_{\mathcal{P}_1})$ whenever $(\mathsf{Fresh}^{\mathsf{n}[\vec{i}]}_{\mathcal{P}_1}(\vec{u}))^{\mathbb{T}}_{\mathcal{P}_1}$ is true.

Our proof system also features a rule expressing the information hiding capabilities of XOR, as well as rules corresponding to the cryptographic assumptions Prf, Cca$_1$, Enc-kp, and Ddh shown in [6].

Finally, our proof system includes a rule FA-Dup which allows to handle some cases where meta-formulas or meta-terms could be dropped from an equivalence, but where this cannot be done using the FA and Dup rules because the

REFL

$$\frac{\vec{u} \text{ is macro-free}}{\Delta \vdash \vec{u} \sim \vec{u}}$$

ENRICH

$$\frac{\Delta \vdash \vec{u}, \xi \sim \vec{v}, \xi'}{\Delta \vdash \vec{u} \sim \vec{v}}$$

DUP

$$\frac{\Delta \vdash \vec{u}, \xi \sim \vec{v}, \xi'}{\Delta \vdash \vec{u}, \xi, \xi \sim \vec{v}, \xi', \xi'}$$

AXIOM

$$\frac{}{\Delta, \vec{u} \sim \vec{v} \vdash \vec{u} \sim \vec{v}}$$

FA

$$\frac{\Delta \vdash \vec{u}, t_1, \ldots, t_n \sim \vec{v}, t'_1, \ldots, t'_n}{\Delta \vdash \vec{u}, \mathsf{f}[\vec{i}](t_1, \ldots, t_n) \sim \vec{v}, \mathsf{f}[\vec{i}](t'_1, \ldots, t'_n)}$$

FA-$\Diamond$

$$\frac{\Delta \vdash \vec{u}, \phi, \phi' \sim \vec{v}, \psi, \psi'}{\Delta \vdash \vec{u}, \phi \Diamond \phi' \sim \vec{v}, \psi \Diamond \psi'} \quad \text{where } \Diamond \in \{\wedge, \vee, \Rightarrow\}$$

EQUIV-TERM

$$\frac{\vdash_{\mathcal{P}_1} t = t' \qquad \Delta \vdash \vec{u}\{t \mapsto t'\} \sim \vec{v}}{\Delta \vdash \vec{u} \sim \vec{v}}$$

EQUIV-FORM

$$\frac{\vdash_{\mathcal{P}_1} \phi \Leftrightarrow \phi' \qquad \Delta \vdash \vec{u}\{\phi \mapsto \phi'\} \sim \vec{v}}{\Delta \vdash \vec{u} \sim \vec{v}}$$

INDUCTION

$$\frac{\Delta \vdash (\vec{u} \sim \vec{v})\{\tau \mapsto \mathsf{init}\} \qquad \{\Delta, (\vec{u} \sim \vec{v})\{\tau \mapsto \mathsf{pred}(\mathsf{a}[\vec{i}])\} \vdash (\vec{u} \sim \vec{v})\{\tau \mapsto \mathsf{a}[\vec{i}]\}\}_{\mathsf{a} \in \mathcal{A}, \vec{i} \notin \mathsf{fv}(\Delta, \vec{u}, \vec{v})}}{\Delta \vdash \vec{u} \sim \vec{v}} \tau \notin \mathsf{fv}(\Delta)$$

Fig. 5. Generic inference rules for equivalences

Base logic rule:

$$\frac{\Delta \vdash \vec{u} \sim \vec{v}}{\Delta \vdash \vec{u}, \mathsf{n} \sim \vec{v}, \mathsf{m}} \quad \text{where } \mathsf{n} \notin \mathsf{st}(\vec{u}),\ \mathsf{m} \notin \mathsf{st}(\vec{v})$$

Meta-logic rule:

$$\frac{\begin{array}{rl}\Delta \vdash & \vec{u}, \mathsf{if}\ \mathsf{Fresh}^{\mathsf{n}[\vec{i}]}_{\mathcal{P}_1}(\vec{u})\ \mathsf{then}\ \mathsf{empty}\ \mathsf{else}\ \mathsf{n}[\vec{i}] \\ \sim & \vec{v}, \mathsf{if}\ \mathsf{Fresh}^{\mathsf{m}[\vec{j}]}_{\mathcal{P}_2}(\vec{v})\ \mathsf{then}\ \mathsf{empty}\ \mathsf{else}\ \mathsf{m}[\vec{j}]\end{array}}{\Delta \vdash \vec{u}, \mathsf{n}[\vec{i}] \sim \vec{v}, \mathsf{m}[\vec{j}]}$$

$$\text{where } \mathsf{Fresh}^{\mathsf{n}[\vec{i}]}_{\mathcal{P}}(\vec{t}) \overset{\text{def}}{=} \bigwedge_{(\mathsf{n}[\vec{i}_0], \vec{j}, c) \in \bar{\mathsf{st}}_{\mathcal{P}}(\vec{t})} \forall \vec{j}. (c \Rightarrow \vec{i} \neq \vec{i}_0)$$

Fig. 6. Rule FRESH.

justification for dropping the elements is not immediately apparent in the meta-logic. Formally, we define in [6] a set of *honest* meta-formulas $\mathsf{Honest}_{\{T\}}$ for any timestamp T. The rule is then as follows:

FA-DUP

$$\frac{\Delta \vdash \vec{u}, \mathsf{frame}@T \sim \vec{v}, \mathsf{frame}@T}{\begin{array}{rl}\Delta \vdash & \vec{u}, \mathsf{frame}@T, \mathsf{exec}@T \wedge \phi \\ \sim & \vec{v}, \mathsf{frame}@T, \mathsf{exec}@T \wedge \phi\end{array}} \ \phi \in \mathsf{Honest}_{\{T\}}$$

Intuitively, we can remove ϕ when it can be computed by the attacker using the information they obtained from a past execution. Our set Honest_S captures a fragment that has this property. Typically, FA-DUP can be used on formulas describing honest interactions between protocol participants, such as the right-hand side of the implication of Listing 2 which does belong to $\mathsf{Honest}_{\{\mathsf{a}_R[j,i]\}}$.

C. Implementation Details

Since we are interested in proving observational equivalence between protocols that are compatible and whose terms only differ in a few positions, we use the common technique of describing two protocols as a single bi-protocol using a special binary function symbol $\mathsf{diff}(\cdot,\cdot)$. Similarly, meta-equivalences are presented as bi-frames, i.e. as a single sequence of meta-terms and formulas featuring the $\mathsf{diff}(\cdot,\cdot)$ symbol. We have derived tactics from the rules presented before, adapting them to work conveniently using this presentation.

VII. CASE STUDIES

We have implemented the meta-logic inside an interactive protocol prover: SQUIRREL. This tool consists of about 10,000 lines of OCaml code, and is available at [50].

SQUIRREL accepts models written in a variant of the applied pi-calculus, as depicted in Listing 1, and allows to prove reachability and equivalence properties of the specified protocol. Proofs are interactive, using tactics. Automated tactics are available to reason over equalities and disequalities over terms (modulo e.g. the equations of XOR). Some tactics performing basic proof search and automated reasoning are implicitly applied at each step of the proof to simplify goals and close absurd ones. A strength of the tool is its modularity: extending the tool with new cryptographic primitives does not impact the core of the tool. It only requires to add new tactics and to prove their soundness inside the meta-logic.

We have used our tool to perform a number of case studies, proving different kind of properties (authentication, strong secrecy, anonymity, unlinkability) under various cryptographic assumptions (PRF, EUF-CMA, XOR, CCA$_1$, ENC-KP, INT-CTXT, DDH). They are summarized in Table I.

For each protocol, we provide the number of lines of code (LoC), the cryptographic assumptions used, and the security properties studied. In all cases, SQUIRREL is able to conclude in less than one minute. Interestingly, most proofs follow the intuition of the pen-and-paper proofs, while some low-level reasoning is successfully abstracted away or automated.

A. RFID based protocols

Those case studies (Basic Hash [23], Hash Lock [40], LAK with pairs instead of XOR as in [39], MW [47] and Feldhofer [38]) are authentication protocols between identity-specific tags and a generic reader (having access to a shared database with authorized tags credentials). We used our tool to establish proofs of unlinkability using the notion defined in [4], [7]: an outside observer must not be able to distinguish

Protocol	**LoC**	**Assumptions**	**Security properties**
Basic Hash [23]	100	PRF, EUF-CMA	Authentication & Unlinkability
Hash Lock [40]	130	PRF, EUF-CMA	Authentication & Unlinkability
LAK (with pairs) [39]	250	PRF, EUF-CMA	Authentication & Unlinkability
MW [47]	300	PRF, EUF-CMA, XOR	Authentication & Unlinkability
Feldhofer [38]	270	ENC-KP, INT-CTXT	Authentication & Unlinkability
Private Authentication [11]	100	CCA$_1$, ENC-KP	Anonymity
Signed DDH [1, ISO 9798-3]	240	EUF-CMA, DDH	Authentication & Strong Secrecy
Additional case studies, using the composition framework from [27]			
Signed DDH [1, ISO 9798-3]	200	EUF-CMA, DDH	Authentication & Strong Secrecy
SSH (with forwarding agent) [51]	700	EUF-CMA, INT-CTXT, DDH	Authentication & Strong Secrecy

TABLE I
CASE STUDIES

between a system where each tag can play many sessions from a system where each tag can play at most one session.

These proofs follow the same global pattern: we use the induction tactic to reason on an arbitrary number of sessions, then at some point we use the equivalent tactic to transform the conditional of an action into an equivalent formula that can be removed with FA-DUP. The systematic use of authentication to establish unlinkability is reminiscent of the well-authentication condition of [7], [39].

In our framework, equivalence requires synchronized executions of the two protocols, but their databases need not have similar structures, as would be the case with the diff-equivalence notions of ProVerif or Tamarin. This has allowed us to obtain proofs of unlinkability that are out-of-scope of these tools, cf. discussion about Basic Hash in [7].

For LAK, MW and Feldhofer protocols, the last conditional of the protocol is not modelled. We managed to partially overcome this limitation for the MW protocol: in this proof, we enrich the frame with infinite sequences of messages that over-approximate what the attacker may learn during protocol executions, which eases the proof process. We cannot yet prove the indistinguishability of these sequences in the tool, because we lack a notion of induction over sequences of messages. This is left for future work.

B. Private authentication

We study the Private Authentication protocol as presented in [11], where the authors give a (manual) proof of anonymity in the computational model for one session of the protocol. A protocol preserves anonymity if an attacker cannot tell whether a session is initiated by one identity or by another. Using our tool, we established a mechanized proof of anonymity for this protocol, for an arbitrary number of sessions.

C. DDH based protocols

We first study a proof of strong secrecy of the shared key for the signed DDH protocol [1, ISO 9798-3]. Similarly to RFID based protocols, we rely on the proof of authentication properties in the reachability prover to perform our proof of strong secrecy in the indistinguishability prover.

We also present two additional case studies for the signed DDH protocol [1, ISO 9798-3] and the SSH protocol [51], where proofs are performed through the use of the composition framework developed for the CCSA model in [27]. The authors outline how to decompose a proof for those protocols into single session proofs, which consists in slightly modifying the hash function by giving more capacities to the attacker using oracles. With our tool, we were able to mechanize those proofs. Compared to the other case studies presented so far, those two hold for an unbounded number of sessions that may depend on the security parameter (this is given by the result in [27]), and not just for an arbitrary number of sessions.

VIII. Conclusion

We have designed a meta-logic on top of the CCSA logic of [11], and proof systems for establishing reachability and equivalence properties in that language. We have shown that it yields a simple, high-level methodology for carrying out computer-assisted proofs of cryptographic protocols, providing asymptotic security guarantees in the computational model. This is supported by the implementation of the interactive prover Squirrel, and its use on various case studies.

As future work, we plan to enrich and optimize the proof automation, for instance by borrowing from the SMT techniques and tools. Our proof systems could also be enriched to benefit from equivalence reasoning in reachability goals, e.g. the PRF equivalence rule should allow to replace hashes by fresh names in reachability goals. We would like also to extend our framework to deal with protocols with states. Whereas handling protocols with states seems to be difficult in CryptoVerif, our approach based on a reasoning over execution traces of protocols seems to be more suitable for this extension. A first application would be, e.g., the mechanization of the modified AKA protocol of [41]. On a more theoretical level, we plan to address the formal semantics of our applied pi-calculus protocols, and the study of their translations to our internal representation as sets of actions. Finally, we plan to elaborate on our approach to provide truly unbounded security guarantees, instead of asymptotic security guarantees for each trace (with an asymptotic bound that may depend on the trace).

Acknowledgment

We would like to thank Bruno Blanchet and Benjamin Grégoire for their interactions and helpful guidance.

References

[1] ISO/IEC 9798-3:2019, IT Security techniques – Entity authentication – Part 3: Mechanisms using digital signature techniques.

[2] Martín Abadi, Bruno Blanchet, and Cédric Fournet. The applied pi calculus: Mobile values, new names, and secure communication. *J. ACM*, 65(1):1:1–1:41, 2018.

[3] Martín Abadi and Phillip Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *J. Cryptology*, 15(2):103–127, 2002.

[4] Myrto Arapinis, Tom Chothia, Eike Ritter, and Mark Ryan. Analysing unlinkability and anonymity using the applied pi calculus. In *CSF*, pages 107–121. IEEE Computer Society, 2010.

[5] Alessandro Armando, David Basin, Yohan Boichut, Yannick Chevalier, Luca Compagna, Jorge Cuéllar, P Hankes Drielsma, Pierre-Cyrille Héam, Olga Kouchnarenko, Jacopo Mantovani, et al. The AVISPA tool for the automated validation of internet security protocols and applications. In *CAV*, pages 281–285. Springer, 2005.

[6] David Baelde, Stéphanie Delaune, Charlie Jacomme, Adrien Koutsos, and Solène Moreau. An interactive prover for protocol verification in the computational model. https://hal.archives-ouvertes.fr/hal-03172119. Technical report.

[7] David Baelde, Stéphanie Delaune, and Solène Moreau. A method for proving unlinkability of stateful protocols. In *CSF*, pages 169–183. IEEE, 2020.

[8] Gergei Bana, Pedro Adão, and Hideki Sakurada. Computationally complete symbolic attacker in action. In *FSTTCS*, volume 18 of *LIPIcs*, pages 546–560. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012.

[9] Gergei Bana, Rohit Chadha, and Ajay Kumar Eeralla. Formal analysis of vote privacy using computationally complete symbolic attacker. In *ESORICS (2)*, volume 11099 of *LNCS*, pages 350–372. Springer, 2018.

[10] Gergei Bana and Hubert Comon-Lundh. Towards unconditional soundness: Computationally complete symbolic attacker. In *POST*, volume 7215 of *Lecture Notes in Computer Science*, pages 189–208. Springer, 2012.

[11] Gergei Bana and Hubert Comon-Lundh. A computationally complete symbolic attacker for equivalence properties. In *CCS*, pages 609–620. ACM, 2014.

[12] Manuel Barbosa, Gilles Barthe, Karthikeyan Bhargavan, Bruno Blanchet, Cas Cremers, Kevin Liao, and Bryan Parno. SoK: Computer-aided cryptography. *IACR Cryptol. ePrint Arch.*, 2019:1393, 2019.

[13] Gilles Barthe, Benjamin Grégoire, Sylvain Heraud, and Santiago Zanella Béguelin. Computer-aided security proofs for the working cryptographer. In *CRYPTO*, volume 6841 of *Lecture Notes in Computer Science*, pages 71–90. Springer, 2011.

[14] David Basin, Jannik Dreier, Lucca Hirschi, Saša Radomirovic, Ralf Sasse, and Vincent Stettler. A formal analysis of 5G authentication. In *CCS*, pages 1383–1396, 2018.

[15] David A. Basin, Cas J. F. Cremers, and Simon Meier. Provably repairing the ISO/IEC 9798 standard for entity authentication. In *POST*, volume 7215 of *Lecture Notes in Computer Science*, pages 129–148. Springer, 2012.

[16] David A. Basin, Jannik Dreier, and Ralf Sasse. Automated symbolic proofs of observational equivalence. In *CCS*, pages 1144–1155. ACM, 2015.

[17] David A. Basin, Andreas Lochbihler, and S. Reza Sefidgar. CryptHOL: Game-based proofs in higher-order logic. *J. Cryptology*, 33(2):494–566, 2020.

[18] Karthikeyan Bhargavan, Bruno Blanchet, and Nadim Kobeissi. Verified models and reference implementations for the TLS 1.3 standard candidate. In *2017 IEEE Symposium on Security and Privacy*, pages 483–502. IEEE, 2017.

[19] Karthikeyan Bhargavan, Barry Bond, Antoine Delignat-Lavaud, Cédric Fournet, Chris Hawblitzel, Catalin Hritcu, Samin Ishtiaq, Markulf Kohlweiss, Rustan Leino, Jay R. Lorch, Kenji Maillard, Jianyang Pan, Bryan Parno, Jonathan Protzenko, Tahina Ramananandro, Ashay Rane, Aseem Rastogi, Nikhil Swamy, Laure Thompson, Peng Wang, Santiago Zanella Béguelin, and Jean Karim Zinzindohoue. Everest: Towards a verified, drop-in replacement of HTTPS. In *SNAPL*, volume 71 of *LIPIcs*, pages 1:1–1:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[20] Bruno Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *CSFW*, pages 82–96. IEEE Computer Society, 2001.

[21] Bruno Blanchet. A computationally sound mechanized prover for security protocols. In *IEEE Symposium on Security and Privacy*, pages 140–154. IEEE Computer Society, 2006.

[22] Bruno Blanchet, Martín Abadi, and Cédric Fournet. Automated verification of selected equivalences for security protocols. In *LICS*, pages 331–340. IEEE Computer Society, 2005.

[23] Mayla Brusò, Konstantinos Chatzikokolakis, and Jerry den Hartog. Formal verification of privacy for RFID systems. In *CSF*, pages 75–88. IEEE Computer Society, 2010.

[24] Ran Canetti, Alley Stoughton, and Mayank Varia. Easyuc: Using easycrypt to mechanize proofs of universally composable security. In *CSF*, pages 167–183. IEEE, 2019.

[25] Rohit Chadha, Vincent Cheval, Ştefan Ciobâcă, and Steve Kremer. Automated verification of equivalence properties of cryptographic protocols. *TOCL*, 17(4):1–32, 2016.

[26] Vincent Cheval, Steve Kremer, and Itsaka Rakotonirina. The DEEPSEC prover. In *CAV*, volume 10982 of *Lecture Notes in Computer Science*, pages 28–36. Springer, 2018.

[27] Hubert Comon, Charlie Jacomme, and Guillaume Scerri. Oracle simulation: a technique for protocol composition with long term shared secrets. In *CCS*, pages 1427–1444, 2020.

[28] Hubert Comon and Adrien Koutsos. Formal computational unlinkability proofs of RFID protocols. In *CSF*, pages 100–114. IEEE Computer Society, 2017.

[29] Hubert Comon-Lundh, Véronique Cortier, and Guillaume Scerri. Tractable inference systems: An extension with a deducibility predicate. In *CADE*, volume 7898 of *Lecture Notes in Computer Science*, pages 91–108. Springer, 2013.

[30] Hubert Comon-Lundh and Stéphanie Delaune. The finite variant property: How to get rid of some algebraic properties. In *RTA*, volume 3467 of *Lecture Notes in Computer Science*, pages 294–307. Springer, 2005.

[31] Véronique Cortier, Constantin Catalin Dragan, François Dupressoir, Benedikt Schmidt, Pierre-Yves Strub, and Bogdan Warinschi. Machine-checked proofs of privacy for electronic voting protocols. In *IEEE Symposium on Security and Privacy*, pages 993–1008. IEEE Computer Society, 2017.

[32] Véronique Cortier, Niklas Grimm, Joseph Lallemand, and Matteo Maffei. Equivalence properties by typing in cryptographic branching protocols. In *International Conference on Principles of Security and Trust*, pages 160–187. Springer, Cham, 2018.

[33] Véronique Cortier, Steve Kremer, and Bogdan Warinschi. A survey of symbolic methods in computational analysis of cryptographic systems. *J. Autom. Reasoning*, 46(3-4):225–259, 2011.

[34] Véronique Cortier and Bogdan Warinschi. A composable computational soundness notion. In *CCS*, pages 63–74. ACM, 2011.

[35] Cas Cremers, Marko Horvat, Jonathan Hoyland, Sam Scott, and Thyla van der Merwe. A comprehensive symbolic analysis of TLS 1.3. In *CCS*, pages 1773–1788, 2017.

[36] Danny Dolev and Andrew Chi-Chih Yao. On the security of public key protocols (extended abstract). In *FOCS*, pages 350–357. IEEE Computer Society, 1981.

[37] Santiago Escobar, Catherine Meadows, and José Meseguer. Maude-NPA: Cryptographic protocol analysis modulo equational properties. In *Foundations of Security Analysis and Design V*, pages 1–50. Springer, 2009.

[38] Martin Feldhofer, Sandra Dominikus, and Johannes Wolkerstorfer. Strong authentication for RFID systems using the AES algorithm. In *CHES*, volume 3156 of *Lecture Notes in Computer Science*, pages 357–370. Springer, 2004.

[39] Lucca Hirschi, David Baelde, and Stéphanie Delaune. A method for unbounded verification of privacy-type properties. *J. Comput. Secur.*, 27(3):277–342, 2019.

[40] Ari Juels and Stephen A. Weis. Defining strong privacy for RFID. *ACM Trans. Inf. Syst. Secur.*, 13(1):7:1–7:23, 2009.

[41] Adrien Koutsos. The 5G-AKA authentication protocol privacy. In *EuroS&P*, pages 464–479. IEEE, 2019.

[42] Adrien Koutsos. Decidability of a sound set of inference rules for computational indistinguishability. In *CSF*, pages 48–61. IEEE, 2019.

[43] Benjamin Lipp, Bruno Blanchet, and Karthikeyan Bhargavan. A mechanised cryptographic proof of the wireguard virtual private network protocol. In *EuroS&P*, pages 231–246. IEEE, 2019.

[44] Gavin Lowe. An attack on the Needham-Schroeder public-key authentication protocol. *Inf. Process. Lett.*, 56(3):131–133, 1995.

[45] Simon Meier, Benedikt Schmidt, Cas Cremers, and David A. Basin. The TAMARIN prover for the symbolic analysis of security protocols. In *CAV*, volume 8044 of *Lecture Notes in Computer Science*, pages 696–701. Springer, 2013.

[46] John C. Mitchell. Multiset rewriting and security protocol analysis. In *RTA*, volume 2378 of *Lecture Notes in Computer Science*, pages 19–22. Springer, 2002.

[47] David Molnar and David A. Wagner. Privacy and security in library RFID: issues, practices, and architectures. In *CCS*, pages 210–219. ACM, 2004.

[48] John Alan Robinson and Andrei Voronkov, editors. *Handbook of Automated Reasoning (in 2 volumes)*. Elsevier and MIT Press, 2001.

[49] Guillaume Scerri and Ryan Stanley-Oakes. Analysis of key wrapping APIs: Generic policies, computational security. In *CSF*, pages 281–295. IEEE Computer Society, 2016.

[50] The Squirrel Prover repository. https://github.com/squirrel-prover/squirrel-prover/.

[51] Tatu Ylonen and Chris Lonvick. The Secure Shell (SSH) Transport Layer Protocol.

Appendix A
Semantics of our meta-logic

Our translation from the meta-logic to the base logic, is parametrized by the protocol $\mathcal{P} = (\mathcal{P}_\mathcal{A}, <)$ that the meta-logic is meant to describe, as well as the trace model $\mathbb{T}$ under study. The general principle of this translation is presented in Section IV-B and we give below the complete definition of this translation. We start with the interpretation of meta-terms of sort index and timestamp:

$$(i)^{\mathbb{T}}_{\mathcal{P}} = \sigma_\mathcal{I}(i) \qquad (\tau)^{\mathbb{T}}_{\mathcal{P}} = \sigma_\mathcal{T}(\tau) \qquad (\mathsf{init})^{\mathbb{T}}_{\mathcal{P}} = \mathsf{init}$$

$$\begin{aligned}(\mathsf{a}[i_1, \dots, i_p])^{\mathbb{T}}_{\mathcal{P}} &= \mathsf{a}_{\sigma_\mathcal{I}(i_1),\dots,\sigma_\mathcal{I}(i_p)} \\ (\mathsf{pred}(T))^{\mathbb{T}}_{\mathcal{P}} &= \mathsf{pred}_\mathcal{T}((T)^{\mathbb{T}}_{\mathcal{P}})\end{aligned}$$

The interpretation of the macro terms m_{init} and $\{m_{\mathsf{a}[\vec{i}]} \mid \mathsf{a}[\vec{i}] \in \mathcal{P}_\mathcal{A}\}$ is already given in Fig. 2. Then, we give the interpretation of meta-terms of sort message:

$$\begin{aligned}
&(\mathsf{n}[i_1, \dots, i_p])^{\mathbb{T}}_{\mathcal{P}} \;=\; \mathsf{n}_{\sigma_\mathcal{I}(i_1),\dots,\sigma_\mathcal{I}(i_p)} \\
&(x)^{\mathbb{T}}_{\mathcal{P}} \;=\; x \\
&(\mathsf{f}[i_1, \dots, i_p](t_1, \dots, t_n))^{\mathbb{T}}_{\mathcal{P}} \;=\; \\
&\qquad\qquad \mathsf{f}_{\sigma_\mathcal{I}(i_1),\dots,\sigma_\mathcal{I}(i_p)}\big((t_1)^{\mathbb{T}}_{\mathcal{P}}, \dots, (t_n)^{\mathbb{T}}_{\mathcal{P}}\big) \\
&(m@T)^{\mathbb{T}} = \begin{cases} (m_{\mathsf{init}})^{\mathbb{T}} & \text{if } (T)^{\mathbb{T}} = \mathsf{init} \\ (m_{\mathsf{a}[\vec{i}]})^{\mathbb{T}\{\vec{i}\mapsto\vec{k}\}} & \text{if } (T)^{\mathbb{T}} = \mathsf{a}[\vec{k}] \text{ and } \mathsf{a}[\vec{i}] \in \mathcal{P}_\mathcal{A} \end{cases} \\
&(\mathsf{if}\ \phi\ \mathsf{then}\ t\ \mathsf{else}\ t')^{\mathbb{T}}_{\mathcal{P}} \;=\; \mathsf{if}\ (\phi)^{\mathbb{T}}_{\mathcal{P}}\ \mathsf{then}\ (t)^{\mathbb{T}}_{\mathcal{P}}\ \mathsf{else}\ (t')^{\mathbb{T}}_{\mathcal{P}} \\
&(\mathsf{find}\ \vec{i}\ \mathsf{suchthat}\ \phi\ \mathsf{in}\ t\ \mathsf{else}\ t')^{\mathbb{T}}_{\mathcal{P}} \;=\; \\
&\qquad\qquad \mathsf{if}\ (\phi)^{\mathbb{T}\{\vec{i}\mapsto\vec{k}_1\}}_{\mathcal{P}}\ \mathsf{then}\ (t)^{\mathbb{T}\{\vec{i}\mapsto\vec{k}_1\}}_{\mathcal{P}}\ \mathsf{else} \\
&\qquad\qquad\qquad \dots \\
&\qquad\qquad \mathsf{if}\ (\phi)^{\mathbb{T}\{\vec{i}\mapsto\vec{k}_p\}}_{\mathcal{P}}\ \mathsf{then}\ (t)^{\mathbb{T}\{\vec{i}\mapsto\vec{k}_p\}}_{\mathcal{P}}\ \mathsf{else}\ (t')^{\mathbb{T}}_{\mathcal{P}}
\end{aligned}$$

where $\vec{k}_1, \dots, \vec{k}_n$ is a complete enumeration of $\mathcal{D}_\mathcal{I}^{|\vec{i}|}$ in lexicographic order[1].

The interpretation of meta-formulas is quite straightforward. For instance, we have that:

$$(\phi \wedge \phi')^{\mathbb{T}}_{\mathcal{P}} \;=\; (\phi)^{\mathbb{T}}_{\mathcal{P}} \mathbin{\dot\wedge} (\phi')^{\mathbb{T}}_{\mathcal{P}}$$

Other boolean connectives are translated similarly. We let:

$$\begin{aligned}
(i = i')^{\mathbb{T}}_{\mathcal{P}} &= \begin{cases} \mathsf{true} & \text{if } \sigma_\mathcal{I}(i) = \sigma_\mathcal{I}(i') \\ \mathsf{false} & \text{otherwise} \end{cases} \\
(T = T')^{\mathbb{T}}_{\mathcal{P}} &= \begin{cases} \mathsf{true} & \text{if } (T)^{\mathbb{T}}_{\mathcal{P}} = (T')^{\mathbb{T}}_{\mathcal{P}} \\ \mathsf{false} & \text{otherwise} \end{cases} \\
(T \leq T')^{\mathbb{T}}_{\mathcal{P}} &= \begin{cases} \mathsf{true} & \text{if } (T)^{\mathbb{T}}_{\mathcal{P}} \leq (T')^{\mathbb{T}}_{\mathcal{P}} \\ \mathsf{false} & \text{otherwise} \end{cases} \\
(t = t')^{\mathbb{T}}_{\mathcal{P}} &= (t)^{\mathbb{T}}_{\mathcal{P}} \doteq (t')^{\mathbb{T}}_{\mathcal{P}}
\end{aligned}$$

Finally, as already explained, quantifications over indices and timestamps are translated to finite boolean expressions:

$$\begin{aligned}
(\forall i.\ \phi)^{\mathbb{T}}_{\mathcal{P}} &= \dot\bigwedge_{k \in D_\mathcal{I}} (\phi)^{\mathbb{T}\{i\mapsto k\}}_{\mathcal{P}} & (\forall \tau.\ \phi)^{\mathbb{T}}_{\mathcal{P}} &= \dot\bigwedge_{v \in D_\mathcal{T}} (\phi)^{\mathbb{T}\{\tau\mapsto v\}}_{\mathcal{P}} \\
(\exists i.\ \phi)^{\mathbb{T}}_{\mathcal{P}} &= \dot\bigvee_{k \in D_\mathcal{I}} (\phi)^{\mathbb{T}\{i\mapsto k\}}_{\mathcal{P}} & (\exists \tau.\ \phi)^{\mathbb{T}}_{\mathcal{P}} &= \dot\bigvee_{v \in D_\mathcal{T}} (\phi)^{\mathbb{T}\{\tau\mapsto v\}}_{\mathcal{P}}
\end{aligned}$$

Overall, this translation is well-defined because our notion of protocol imposes that the condition and output message of an action α only refer to actions $\beta < \alpha$, or to $\mathsf{input}@\alpha$ (which itself can only refer to actions $\beta < \alpha$). For instance, the translation of $(\mathsf{output}@T)^{\mathbb{T}}$ is defined as the translation of a potentially large term (the output message at $(T)^{\mathbb{T}}$) but its translation can only rely on translations $(m@T')^{\mathbb{T}}$ for $(T')^{\mathbb{T}} < (T)^{\mathbb{T}}$ or $m = \mathsf{input}$ and $(T')^{\mathbb{T}} = (T)^{\mathbb{T}}$.

Appendix B
Reachability Sequent Calculus

We present in this section the soundness proof for the reachability rules of Fig. 3. More complex reachability rules (e.g. Int-ctxt), along with the formal definition of $\bar{\mathsf{st}}_{\mathcal{P}}(\_)$ and their soundness proof, are provided in the long version [6].

Proposition 3. *The rules of Fig. 3 are sound.*

Proof. To prove the rules soundness, we only need to show that, if the premises are valid, then the conclusion is valid. We only show two rules here. The full soundness proof is in the long version [6].

- For NameIndep, this is because names of the meta-logic with different head symbols are always translated as different names of the base logic. We conclude by using the base logic rule which states that $\mathsf{EQ}(\mathsf{n}, \mathsf{m}) \sim \mathsf{false}$ whenever n and m are distinct names (this is the rule EqIndep of [28]).
- For NameEq, if one of the equalities $i_1 = j_1, \dots, i_k = j_k$ does not hold in $\mathbb{T}$, then we know that $(\mathsf{n}[i_1, \dots, i_k])^{\mathbb{T}}$ and $(\mathsf{n}[j_1, \dots, j_k])^{\mathbb{T}}$ are distinct names of the base logic. Again, we conclude using the EqIndep rule of [28]. □

Appendix C
Equivalence Rules

We present in this section the Xor-Fresh rule, and its soundness proof along with the one for the Fresh rule. More advanced rules (CCA_1, DDH and ENC-KP) and their soundness proofs are provided in the long version [6].

[1] This is important to establish the soundness of some tactics, e.g. the one used to remove useless indices of the `try find` command.

Base logic rule:

$$\frac{\Delta \vdash \vec{u} \sim \vec{v}, \big(\mathsf{if}\ \mathsf{len}(t) = \mathsf{len}(\mathsf{n})\ \mathsf{then}\ \mathsf{n}\ \mathsf{else}\ t \oplus \mathsf{n}\big)}{\Delta \vdash \vec{u} \sim \vec{v}, t \oplus \mathsf{n}}$$

where $\mathsf{n} \notin \mathsf{st}(\vec{v}, t)$

Meta-logic rule:

$$\frac{\begin{array}{c}\Delta \vdash \vec{u} \sim \vec{v}, \big(\mathsf{if}\ \mathsf{len}(t) = \mathsf{len}(\mathsf{n}[\vec{i}]) \wedge \mathsf{Fresh}^{\mathsf{n}[\vec{i}]}_{\mathcal{P}_2}(\vec{v}, t) \\ \mathsf{then}\ \mathsf{n}[\vec{i}]\ \mathsf{else}\ t \oplus \mathsf{n}[\vec{i}]\big)\end{array}}{\Delta \vdash \vec{u} \sim \vec{v}, t \oplus \mathsf{n}[\vec{i}]}$$

Fig. 7. Rule XOR-FRESH (base and meta logic).

a) XOR equivalence rule: The Fig. 7 defines our XOR-FRESH rule, which expresses the information hiding capabilities of XOR. Together with the rules of Section V, these are the only rules that we need to reason about protocols involving XOR. In a nutshell, the rule expresses that $t \oplus \mathsf{n}[\vec{i}]$ and $\mathsf{n}[\vec{i}]$ are interpreted as the same probability distribution provided that $\mathsf{n}[\vec{i}]$ is fresh (this guarantees the independence of distributions) and that the distributions yield messages of the same length. In details, we rely on a base logic rule that differs from the one of [28]. We first establish its soundness (see Proposition 5), and then lift it using the same over-approximation of the freshness condition as for FRESH.

b) Soundness: We now prove the soundness of our rules.

Proposition 4. *Rule* FRESH *is sound.*

Proof sketch. Consider an instance of the rule with a valid premise, and conclusion ϕ. Let us show that, for any $\mathbb{T}$ and $\mathbb{M}$ satisfying Δ, we also have $\mathbb{T}, \mathbb{M} \models \phi$. Note that $(\mathsf{Fresh}^{\mathsf{n}[\vec{i}]}_{\mathcal{P}_1}(\vec{u}))^{\mathbb{T}}_{\mathcal{P}_1}$ is a boolean combination of constants true and false, and similarly for the other freshness formula.

It cannot be that the interpretation of one formula is equivalent to true and the other to false, as this would contradict the validity of the premise. If the two interpretations are false, the premise is equivalent to the conclusion, so we conclude trivially.

Otherwise, the two freshness formulas interpret to true. This implies that $\mathsf{n}_{\sigma_{\mathcal{I}}(\vec{i})}$ is not a subterm of $(\vec{u})^{\mathbb{T}}_{\mathcal{P}_1}$, and correspondingly for $\mathsf{m}_{\sigma_{\mathcal{I}}(\vec{j})}$ and $\vec{v}$. The validity of the premise gives us $\mathbb{T}, \mathbb{M} \models (\vec{u})^{\mathbb{T}}_{\mathcal{P}_1}, \mathsf{empty} \sim (\vec{v})^{\mathbb{T}}_{\mathcal{P}_2}, \mathsf{empty}$, which implies $\mathbb{T}, \mathbb{M} \models (\vec{u})^{\mathbb{T}}_{\mathcal{P}_1} \sim (\vec{v})^{\mathbb{T}}_{\mathcal{P}_2}$. We can finally apply the FRESH rule of the base logic to obtain $\mathbb{T}, \mathbb{M} \models \phi$ as expected. □

Proposition 5. *Rule* XOR-FRESH *is sound in all computational models where $\oplus$ is interpreted as XOR.*

Proof. It suffices to show that the base logic rule is sound, then the lifting works as for FRESH. We express the advantage of an attacker on the indistinguishability game of the conclusion, where L is the random variable $[\![\mathsf{len}(t) = \mathsf{len}(\mathsf{n})]\!]$ and t' is $\mathsf{if}\ \mathsf{len}(t) = \mathsf{len}(\mathsf{n})\ \mathsf{then}\ \mathsf{n}\ \mathsf{else}\ t \oplus \mathsf{n}$:

$$\begin{aligned}
&\mathrm{Adv}(\mathcal{A}) \\
&= |\mathbf{Pr}[\mathcal{A}([\![\vec{u}]\!])] - \mathbf{Pr}[\mathcal{A}([\![\vec{v}, t \oplus \mathsf{n}]\!])]| \\
&= |\mathbf{Pr}[L] \times \big(\mathbf{Pr}[\mathcal{A}([\![\vec{u}]\!]) \mid L] - \mathbf{Pr}[\mathcal{A}([\![\vec{v}, t \oplus \mathsf{n}]\!]) \mid L]\big) + \\
&\quad \mathbf{Pr}[\neg L] \times \big(\mathbf{Pr}[\mathcal{A}([\![\vec{u}]\!]) \mid \neg L] - \mathbf{Pr}[\mathcal{A}([\![\vec{v}, t \oplus \mathsf{n}]\!]) \mid \neg L]\big)|
\end{aligned}$$

The freshness condition $\mathsf{n} \notin \mathsf{st}(\vec{u}, t)$ implies that the distributions $[\![\vec{v}, t \oplus \mathsf{n}]\!]$ and $[\![\vec{v}, \mathsf{n}]\!]$ are the same, provided that L holds. Moreover, when $\neg L$, the terms $t \oplus \mathsf{n}$ and t' evaluate to the same result. We can thus rewrite our advantage as follows:

$$\begin{aligned}
&\mathrm{Adv}(\mathcal{A}) \\
&= |\mathbf{Pr}[L] \times \big(\mathbf{Pr}[\mathcal{A}([\![\vec{u}]\!]) \mid L] - \mathbf{Pr}[\mathcal{A}([\![\vec{v}, t']\!]) \mid L]\big) + \\
&\quad \mathbf{Pr}[\neg L] \times \big(\mathbf{Pr}[\mathcal{A}([\![\vec{u}]\!]) \mid \neg L] - \mathbf{Pr}[\mathcal{A}([\![\vec{v}, t']\!]) \mid \neg L]\big)| \\
&= |\mathbf{Pr}[\mathcal{A}([\![\vec{u}]\!])] - \mathbf{Pr}[\mathcal{A}([\![\vec{v}, t']\!])]|
\end{aligned}$$

In other words the advantage is the same for the premise and conclusion, thus the rule is sound. □

APPENDIX D
NUMBER OF SESSIONS DEPENDING ON η

We give here an example of two protocols P and $P_{\mathcal{I}}$ that can be distinguished by an adversary which can interact $q(\eta)$ times with the protocol, where q is a polynomial, but cannot be distinguished by an adversary which interacts k times with the protocol (where k is an arbitrary fixed integer, independent of η). The construction of P is as follows. First, the protocol samples uniformly at random a bitstring n of length η. Then, when queried with an integer input i, the protocol leaks $\mathsf{bit}(\mathsf{n}, \mathsf{i})$ – the i^{th} bit of the nonce n. Finally, the adversary wins the game if they can find n. The idealised protocol $P_{\mathcal{I}}$ is identical to P, except that the adversary never wins:

$$P_{\mathsf{bit}}(\mathsf{n}) \ := \ !\ (\mathsf{in}(c_1, \mathsf{i}); \mathsf{out}(c_2, \mathsf{bit}(\mathsf{n}, \mathsf{i})))$$

$$P := \mathsf{new}\ \mathsf{n}; \left(\begin{array}{r} P_{\mathsf{bit}}(\mathsf{n}) \mid \mathsf{in}(c_3, \mathsf{x}); \mathsf{if}\ \mathsf{n} = \mathsf{x}\ \mathsf{then}\ \mathsf{out}(c_4, \mathsf{true}) \\ \mathsf{else}\ \mathsf{out}(c_4, \mathsf{false}) \end{array}\right)$$

$$P_{\mathcal{I}} := \mathsf{new}\ \mathsf{n}; \big(P_{\mathsf{bit}}(\mathsf{n}) \mid \mathsf{in}(c_3, \mathsf{x}); \mathsf{out}(c_4, \mathsf{false}\big)$$

Clearly, there exists an adversary $\mathcal{A}$ that can distinguish between P and $P_{\mathcal{I}}$ by interacting $\eta{+}1$ times with the protocol: this adversary queries all bits of n using η queries, and then sends n to the protocol. In that scenario, P returns true while $P_{\mathcal{I}}$ returns false, which are trivial to distinguish.

Of course, an adversary interacting at most k times with the protocol cannot learn more than k bits of n, and has a winning probability of at most $2^{k-\eta}$. Such an adversary has a negligible probability of winning (w.r.t. η).

APPENDIX E
COMPARISON WITH EXISTING TOOLS

To deepen the comparison between SQUIRREL, CRYPTOVERIF and EASYCRYPT, we conducted a security analysis of the same protocol in the three tools.[2] More precisely, we modelled the Basic Hash protocol, and proved that it

[2] Of course, we also used existing case studies in the literature (e.g. [28], [31], [43]), albeit not of the same protocol, to build a detailed description of how each tool operates.

provides authentication and unlinkability. The corresponding SQUIRREL, CRYPTOVERIF and EASYCRYPT files can be found in the repository [50].[3]

We start by presenting the summary of the key findings of our comparison in Appendix E-A. This summary is based on the basic quantitative comparison of the three approaches in Appendix E-B, and an in-depth qualitative comparison conducted in Appendices E-C, to E-F, where we compare the three approaches on how protocols, security properties and cryptographic assumptions are modelled, and on how security proofs are carried out.

A. Summary of key findings

Generally, we conclude that CRYPTOVERIF and SQUIRREL operates at a similar level of details and expressivity, though they use different approaches, and proofs developments. On the other hand, EASYCRYPT is more expressive, at the cost of a higher level of details and modelling overhead.

a) Protocols: Protocols are modelled in very similar fashion in CRYPTOVERIF and SQUIRREL. Writing a protocol in EASYCRYPT may require a more involved modelling – because it does not support directly protocols with sequences of inputs/outputs.

We note that CRYPTOVERIF does not support stateful protocols, while EASYCRYPT does. The CCSA approach on which SQUIRREL is based upon can naturally model stateful protocols [41] – such an extension is left as future work.

b) Security properties: From our case study, the three tools appeared equally capable of expressing all usual security properties, albeit using different approaches. However, we note that: i) native support for events or timestamps in CRYPTOVERIF and SQUIRREL allows for more direct and simpler statements of correspondence properties; and ii) EASYCRYPT logic is more expressive, and allows to formalize more complex relational properties of protocols, and to internalize composition results and reasoning (though we did not much exploit this in our case study).

c) Cryptographic assumptions: Adding new cryptographic assumptions in EASYCRYPT is easy, as it uses a very expressive logic. While CRYPTOVERIF provides a way to add new assumptions, doing it in a way that can be efficiently used by the tool probably requires a deep understanding of the tool. On the other side of the spectrum, we have SQUIRREL, where adding new hypotheses is not possible without in-depth knowledge of the tool.

d) Proofs: Proofs in CRYPTOVERIF and SQUIRREL mostly focus on the security aspects, thanks to native support for cryptographic reasoning, although using different approaches: the former uses cryptographic game transformation, while the latter operates on the protocol traces. On the other hand, EASYCRYPT does not have rules dedicated to cryptographic reasoning, but allows to carry out such reasoning using its lower-level, very expressive logics. As a direct consequence, proofs are considerably shorter in CRYPTOVERIF and SQUIRREL than in EASYCRYPT.

	SQUIRREL	CRYPTOVERIF	EASYCRYPT
Authentication			
- modelling	30 LoC	40 LoC	220 LoC
- proof	10 LoC	0 LoC	60 LoC
Unlinkability			
- modelling	20 LoC	60 LoC	330 LoC
- proof	40 LoC	10 LoC	630 LoC

TABLE II
TOOLS COMPARISON: BASIC HASH PROTOCOL

B. Quantitative Comparison

As a basic quantitative metric, we give in Table II the lines of code (LoC) needed to prove the authentication and unlinkability properties of the Basic Hash protocol in each tool, distinguishing the modelling and proof parts. We notice that SQUIRREL and CRYPTOVERIF have similar LoC values, with the difference that CRYPTOVERIF concludes automatically for the authentication proof. The EASYCRYPT development is sensibly longer, which was expected considering the fact that it is a general purpose proof assistant, relying on an expressive high-order logic.

We stress the fact that the LoC metric only gives a very rough idea of the modelling and proof efforts required to perform our case study analysis. For example, a large part of the EASYCRYPT development are low-level Hoare-logic proofs which, while being protracted, are elementary.

C. Qualitative comparison: modelling protocols

In CRYPTOVERIF, protocols are modelled using a variant of the applied-pi calculus. This language is similar to the input language of SQUIRREL. We note however that SQUIRREL can model trace restrictions, which can be used, for example, to model protocol phases (see Example 6). Phases and trace restrictions are not supported by CRYPTOVERIF.

EASYCRYPT has been mostly used for security analysis of cryptographic primitives. These are modelled using modules, which are simply lists of procedures. These procedures operate on a global memory and can perform random samplings – they are stateful probabilistic programs. A key feature of the module system is its composability: a module $\mathcal{F}$ – a functor – can be parameterized by other modules by taking them as arguments. For example, an encryption schema can be defined w.r.t. an unspecified block permutation. The same mechanism can be used for protocols, modelling an input/output pair as a module procedure. Note however that when modelling a protocol in which an agent performs sequentially multiple pairs of input/output, the modelling becomes more complex, as we need to maintain in a global state the position of the agent in this sequence (this problem does not appear in Basic Hash, as each agent comprises only one input/output pair).

D. Qualitative Comparison: modelling security properties

a) Correspondence Properties: In CRYPTOVERIF, the processes of the protocol agents can be annotated by events. Crucially, events do not modify the agents behavior: they are

[3] See `examples/{README.md, cryptoverif/, easycrypt/}`

only added to the execution trace of the protocol, where they can be used to express properties of the protocol.

Modelling correspondence properties in SQUIRREL is very similar, though we do not use events: we directly refer to the values of components of the protocol (e.g. nonces, messages) at various points of the protocol execution using timestamps.

Expressing correspondence properties in EASYCRYPT requires more work. For the Basic Hash protocol, we had to modify the protocol agents to add some bookkeeping of the events: essentially, events are manually logged into some global tables. Security properties are then expressed in the same way as in CRYPTOVERIF or SQUIRREL.

Contrarily to our timestamp approach where nothing is added, and to CRYPTOVERIF events that are just annotations leaving the protocol behavior unchanged, the EASYCRYPT modelling of correspondence properties has the drawback that it requires to modify the protocol to add the global tables of events: the protocol shown secure is not the original protocol. Of course, this problem can be solved by doing the modelling in several steps: i) define the original protocol; ii) define the protocol with events; and iii) show that the two protocols have the same behavior (except for the global tables). Nonetheless, this requires some additional work from the user.

Arguably, a minor advantage of our approach compared to CRYPTOVERIF is that it allows to have more concise protocol descriptions, as we do not need to modify the protocol specification itself. For example, new correspondence properties can be added to a development without changing the protocol.

b) Equivalence properties: CRYPTOVERIF models equivalence properties as the indistinguishability of two protocols (given as two distinct processes), whereas in SQUIRREL the two protocols have to be given in a bi-process.

EASYCRYPT models equivalences using relational probabilistic Hoare logic formulas (pRHL), which state that, assuming that the initial memories of the left and right programs satisfy some relational property (the pre-condition), the final memories satisfy some other relational property (the post-condition). This is a very expressive logic, which goes beyond bare equivalence: e.g., we can write a pRHL formula that (roughly) states that an event happened on the left program *less often* than on the right program.

E. Modelling: cryptographic assumptions

The analyses have been carried out using the same cryptographic assumptions: the keyed hash function is assumed EUF-CMA and PRF. Although the cryptographic assumptions are the same, their modelling differs.

CRYPTOVERIF offers a large panel of predefined cryptographic primitives: when modelling a protocol, a user has to import the appropriate cryptographic primitives and assumptions they need. Notably, these are modelled using a specification language, which let the user declares sets of indistinguishable oracles available to the adversary. While an experienced user can theoretically use this system to define their own cryptographic assumptions, we remark that some of the default assumptions have been modified in non-trivial ways, probably to use alternative equivalent formulations that are more powerful and can be used automatically by the tool. Consequently, adding complex new assumptions may be out-of-reach of most users.

In EASYCRYPT, cryptographic assumptions are expressed through games, using the same programming language and module system than for protocols, and the user can define new cryptographic hypotheses if needed. Moreover, this can be done easily, as the language used to express assumptions is very close to the pseudo-code language used by cryptographers for paper proofs.

SQUIRREL cryptographic hypotheses are hard-coded in the prover, and adding now cryptographic hypothesis is not straightforward: it requires to do non-trivial syntactic checks, and to compute complex sets of subterms. E.g., for the EUF-CMA, we must compute the set of hashes appearing in a protocol execution for a given key, and the condition on the trace under which each of these hashes appears.

F. Security proofs

The main differences between CRYPTOVERIF and SQUIRREL lie in the way proofs are handled. Each step of the proof is a transformation of a cryptographic game in CRYPTOVERIF, while in SQUIRREL the transformation is applied to formulas and frames. As a first consequence of this difference, CRYPTOVERIF does not handle states, while we are confident that SQUIRREL will be able to support stateful protocols.[4] Both tools have basic proof steps dedicated to cryptographic reasoning: CRYPTOVERIF applies cryptographic assumptions by replacing a set of oracles by an indistinguishable set of oracles, while SQUIRREL has specialized inference rules.

EASYCRYPT proofs are done at two levels. First, the equivalence between two protocols (modelled as modules) can be shown using a probabilistic relational Hoare logic (pRHL). To complete such proofs, the user often has to come up with (relational) invariants of the programs, and to perform precise probabilistic reasoning. Then, such results can be composed using its ambient higher-order logic (e.g. a property shown for an arbitrary block permutation can be instantiated to a precise permutation, say AES). Among these three tools, EASYCRYPT is the only one that allows to prove high-level composition results (e.g. the Universal Composability framework has been formalized in EASYCRYPT [24]). On the other hand, and in contrast with CRYPTOVERIF and SQUIRREL, EASYCRYPT does not have rules dedicated to cryptographic reasoning. Instead, its pRHL and higher-order logic are expressive enough to carry out cryptographic proofs directly – often at the cost of more details and work.

[4] The CCSA approach can be naturally adapted to model and prove secure stateful protocols (e.g. see the manual case study of [41]).