Extending the SQUIRREL meta-logic for reasoning over security protocols

David Baelde, Stéphanie Delaune, Charlie Jacomme, Adrien Koutsos, Solène Moreau

Extending the SQUIRREL meta-logic for reasoning over security protocols*

Work in Progress

David Baelde
LMF, ENS Paris-Saclay & CNRS, Université Paris-Saclay, France

Stéphanie Delaune
Univ Rennes, CNRS, IRISA, France

Charlie Jacomme
CISPA Helmholtz Center for Information Security, Germany

Adrien Koutsos
Inria Paris, France

Solène Moreau
Univ Rennes, CNRS, IRISA, France

The formal verification of security protocols can be carried out in two categories of models. Symbolic models, pioneered by Dolev and Yao, represent messages by first-order terms and attacker capabilities by inference rules or equational theories. This approach allows automatic verification, notably using techniques from rewriting and logic. The computational model, however, represents messages as bitstrings and attackers as probabilistic polynomial-time (PTIME) Turing machines. It is the standard model for provable security in cryptography, but formal verification in that model remains challenging. Bana and Comon have recently proposed a compelling approach to derive guarantees in the computational model, which they call the computationally complete symbolic attacker (CCSA). It is based on first-order logic, with complex axioms derived from cryptographic assumptions. Verification in the original CCSA approach involves a translation of protocol traces into first-order terms that necessitates to bound the length of traces. The generated goals are difficult to process by humans and, so far, cannot be handled automatically either. We have proposed a refinement of the approach based on a meta-logic which conveniently replaces the translation step. We have implemented this refined approach in an interactive theorem prover, SQUIRREL, and validated it on a first set of case studies. In this paper, we present three improvements of the foundations of the SQUIRREL meta-logic and of its proof system, which are required as we are considering more complex case studies. First, we extend our model to support memory cells in order to allow the analysis of security protocols that manipulate states. Second, we adapt the notion of trace model, which provides the semantics of our meta-logic formulas, to enable more natural and expressive modelling. Finally, we present a generalized proof system which allows to derive more protocol properties.

1 Introduction

SQUIRREL [14] is a proof assistant that allows to prove properties of security protocols against arbitrary attackers. It works with standard cryptographic notions: security properties are established against arbitrary attackers modelled as probabilistic polynomial-time computations, and assuming properties of cryptographic primitives such as indistinguishability (IND-CCA), unforgeability (EUF-CMA) or pseudo-randomness (PRF). It thus lies in the same category as CRYPTOVERIF [8], EASYCRYPT [6], or CryptHOL [7], although it is much more recent and less mature than these systems. SQUIRREL implements a meta-logic which is built on top of the logic introduced by Bana and Comon [5], following an approach which they call the computationally complete symbolic attacker (CCSA). The verification

*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.

method proposed by Bana and Comon assumes a fixed bound on protocol traces and relies on an explicit encoding of protocol traces as terms of the CCSA logic. It yields verification goals that are tedious to prove by hand and for which no automated prover exists so far. The meta-logic behind SQUIRREL internalizes the encoding of traces as terms, and allows high-level proofs that are well-suited for interactive theorem proving. Moreover, these proofs hold for any trace, without a bound on their length. SQUIRREL and its meta-logic have enabled the first mechanized proofs of protocols using the CCSA approach [1].

Since its initial presentation [1], the SQUIRREL system has evolved in a number of ways. We present in this paper several improvements to its meta-logic. First, we enrich the considered notion of protocol with the possibility to model mutable states, which is important for many practical applications. Second, we adapt the semantics to more naturally represent execution traces. This enables more precise reasoning on traces, but also the possibility to express constraints on traces that are practically relevant, such as phases. Third, we generalize the kind of sequents considered in our proof systems. This is in part necessary to be able to adapt proofs from the original formalism with the new semantics, but we show that a systematic generalization actually brings more expressivity. The support for states and the move to the new semantics are implemented in the current version of SQUIRREL, and they have been used to carry out new proofs of protocols. However, the generalized sequents presented in this paper are only partially implemented as of today.

Outline. We start by providing some background on the base CCSA logic in Section 2. We then introduce in Section 3 the improved syntax and semantics of our meta-logic. We describe in Section 4 the generalized sequent calculi that we use to reason in our meta-logic. We illustrate their new improvements with a case study in Section 5.

2 Background

The CCSA approach [4, 5] allows to reason about probabilistic polynomial-time computations using the simple symbolic setting of first-order logic. In a nutshell, terms will be used to model computations, and a single predicate is going to model indistinguishability of (sequences of) terms. Standard cryptographic assumptions can then be reformulated as axiom schemes, from which the indistinguishabilities of interest will have to be derived. This approach has been demonstrated on various protocols to obtain formal proofs of security [2, 13, 10, 3, 12]. In this section, we briefly present the logic of [5], to which we will refer to as the *base logic*. In comparison with [5], we consider only the sort message, and a single attacker symbol att.

Example 1 (Running example)**.** *We consider a variant of the OSK protocol in the spirit of the description given in [9], which is an RFID protocol involving tags. Each tag is associated to secret keys k, k' and has a mutable state s initialized with a secret value n. At each session of the protocol, a tag updates s with $\mathsf{H}(s,k)$ and outputs $\mathsf{G}(s,k')$, with H and G two keyed hash functions. Intuitively, readers knowing the tag's secrets will be able to recognize its outputs, but an attacker will not be able to learn these secrets and impersonate tags.*

Syntax of the base logic. 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 (modeling random samplings), a set of variables $\mathcal{X}_B$, and a set of function symbols $\mathcal{F}_B$ (modeling, e.g., cryptographic primitives). 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 $\mathsf{EQ}(\_,\_)$, conditionals $\mathsf{if}\ \_\ \mathsf{then}\ \_\ \mathsf{else}\ \_$;
- constants empty, true and false.

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. Such a formula intuitively expresses that the two sequences of messages are indistinguishable.

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, the term $\mathsf{EQ}(\mathsf{true}, \mathsf{EQ}(u,v))$ or the formula $\mathsf{EQ}(u,v) \sim \mathsf{true}$.

Example 2. *To model the OSK protocol of Example 1, we use the name symbols* n, k *and* k', *and two function symbols* H *and* G. *The term* $\mathsf{G}(\mathsf{H}(n,k),k')$ *represents the message outputted at the first session of the tag. Assuming two different tags with respective secrets* n_1, k_1, k'_1 *and* n_2, k_2, k'_2, *the atomic formula*

$$\mathsf{G}(\mathsf{H}(n_1,k_1),k'_1), \mathsf{G}(\mathsf{H}(n_2,k_2),k'_2) \sim \mathsf{G}(\mathsf{H}(n_1,k_1),k'_1), \mathsf{G}(\mathsf{H}(\mathsf{H}(n_1,k_1),k_1),k'_1),$$

expresses a simple unlinkability property: an outside observer cannot tell the difference between the outputs of two different tags and two successive outputs of the same tag. We can also write an equivalence expressing that an attacker cannot forge the second tag output given the first one, which holds in all models where the interpretation of G *is unforgeable:*

$$\mathsf{EQ}\big(\mathsf{att}(\mathsf{G}(\mathsf{H}(n,k),k')), \mathsf{G}(\mathsf{H}(\mathsf{H}(n,k),k),k')\big) \sim \mathsf{false}$$

Semantics of the base logic. 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 probabilistic PTIME Turing machines that run in polynomial-time w.r.t a security parameter η and that manipulate a pair $\rho = (\rho_s, \rho_r)$ of infinite read-only 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 (this ensure that syntactically distinct names are interpreted as *independent* random variables).
- The symbols empty, true, false, EQ and $\mathsf{if}\ \_\ \mathsf{then}\ \_\ \mathsf{else}\ \_$ are interpreted in the expected way. For instance, 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.
- 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 will put additional restrictions on the computational models we consider, according to the assumptions the protocol relies on: we may assume e.g. that a binary function symbol $\oplus$ is interpreted as exclusive or, or that a binary function symbol H is interpreted as a cryptographically secure 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 $\approx$. Intuitively, two terms are computationally indistinguishable if a probabilistic PTIME machine can only distinguish them with negligible probability. More precisely, we have $d_1,\ldots,d_n \approx d'_1,\ldots,d'_n$ when for any PTIME Turing machine $\mathscr{A}$, the following quantity is negligible in η:

$$|\ \mathbf{Pr}(\rho : \mathscr{A}(d_1(1^\eta,\rho),\ldots,d_n(1^\eta,\rho),\rho_r) = 1) - \mathbf{Pr}(\rho : \mathscr{A}(d'_1(1^\eta,\rho),\ldots,d'_n(1^\eta,\rho),\rho_r) = 1)\ |.$$

Here, $\mathbf{Pr}(\rho : \mathscr{M}(\rho))$ is the probability that the machine $\mathscr{M}$ accepts w.r.t the random tape ρ. As is standard with asymptotic security, a function is said to be negligible when it is asymptotically smaller than the inverse of any polynomial.

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

Example 3. *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, no attacker can distinguish between two random samplings with the same distribution, and there is a negligible probability that two independent uniform samplings of length* η *coincide.*

Proof system for the base logic. In this approach, a proof of a security property is done using an axiomatic approach, by deriving the formula expressing the given security property from a set of sound axioms, using any proof system for first-order logic. Some axioms are sound in all computational models, e.g., the symmetry of $\sim$, or properties of $\mathsf{if\ \_\ then\ \_\ else\ \_}$. Other axioms reflect cryptographic assumptions on the security primitives, such as the *Pseudo Random Function* (PRF [11]) assumption for hash functions [1]. Such axioms exclude any computational model that does not satisfy the assumptions under which a security protocol has been designed (e.g., that the protocol must be instantiated with a secure hash function). The equivalences from Example 2 can for instance be derived using the PRF axiom scheme and axioms about $\mathsf{if\ \_\ then\ \_\ else\ \_}$.

3 Syntax and semantics of our meta-logic

In this section, we introduce the syntax and the semantics of our meta-logic, which can be seen as an extension of the meta-logic we proposed in [1] with memory cells (which allows to model stateful security protocols). In order to match the intuitive notion of a trace, we add $\mathsf{happens}(T)$ as an atomic formula over timestamps to express the fact that an action happens. Actually, our previous definition (the one proposed in [1]) requires that all possible actions were present in the trace, even though they represent e.g. mutually exclusive actions issuing from a conditional. In addition to this change, we also introduce the notion of global meta-formulas to enrich our notion of sequents and to be able to design a more powerful proof system. Altogether, this allows us to express and prove security properties that were out-of-scope of our previous formalism.

3.1 Local meta-logic formulas

Syntax. Our meta-logic is a first-order logic with terms of three possible sorts: *timestamp* to represent time points in an execution; *message* to represent bitstrings exchanged between protocol's participants, or stored in the participants states; and *index* are used to identify an element, e.g. a session or an item in a database. We consider infinite sets $\mathscr{T}$, $\mathscr{X}$, and $\mathscr{I}$ of variables of each sort.

[1] Informally, a keyed hash function $\mathsf{H}(\_,\mathsf{k})$ satisfies the PRF assumption if its output on a message never hashed before is computationally indistinguishable from a value sampled uniformly at random.

$$
\begin{array}{rcll}
T & := & \tau & \textit{(variable, } \tau \in \mathscr{T}\textit{)} \\
 & | & \mathsf{a}[i_1,\ldots,i_k] & \textit{(action, } \mathsf{a} \in \mathscr{A}\textit{)} \\
 & | & \mathsf{init} \mid \mathsf{pred}(T) & \\
t & := & x & \textit{(variable, } x \in \mathscr{X}\textit{)} \\
 & | & s[i_1,\ldots,i_k]@T & \textit{(memory cell, } s \in \mathscr{S}\textit{)} \\
 & | & \mathsf{n}[i_1,\ldots,i_k] & \textit{(name, } \mathsf{n} \in \mathscr{N}\textit{)} \\
 & | & \mathsf{f}[i_1,\ldots,i_k](t_1,\ldots,t_n) & \textit{(function, } \mathsf{f} \in \mathscr{F}\textit{)} \\
 & | & \mathsf{input}@T \mid \mathsf{output}@T \mid \mathsf{frame}@T & \textit{(macros)} \\
 & | & \mathsf{if}\ \phi\ \mathsf{then}\ t\ \mathsf{else}\ t' & \\
 & | & \mathsf{find}\ \vec{i}\ \mathsf{suchthat}\ \phi\ \mathsf{in}\ t\ \mathsf{else}\ t' & \textit{(index lookup)}
\end{array}
$$

$$
\begin{array}{rcll}
i & := & \iota & \textit{(variable, } \iota \in \mathscr{I}\textit{)} \\
A & := & t = t' \mid i = i' \mid T = T' & \\
 & | & T < T' \mid T \leq T' & \\
 & | & \mathsf{happens}(T) & \\
 & | & \mathsf{cond}@T \mid \mathsf{exec}@T & \\
\phi & := & A \mid \top \mid \bot & \\
 & | & \phi \wedge \phi' \mid \phi \vee \phi' \mid \phi \Rightarrow \phi' \mid \neg\phi & \\
 & | & \forall i.\phi \mid \exists i.\phi \mid \forall \tau.\phi \mid \exists \tau.\phi &
\end{array}
$$

Figure 1: Syntax of meta-terms and meta-formulas

We assume a set $\mathscr{F}$ of indexed function symbols to represent protocol functions and cryptographic primitives (e.g encryptions or keyed hash functions). Function symbols representing cryptographic primitives will have index arity 0, and a message arity depending on the primitive. In contrast, function symbols representing identities or keys will have 0 as message arity and 1 (or even more) as index arity. We also assume a set $\mathscr{N}$ of indexed name symbols to model random samplings.

Then, we have a set of indexed action symbols $\mathscr{A}$ which are used to model specific time points. To model protocol messages as terms, we use macros. Essentially, a macro is a term construct that represents a specific bitstring value of the protocol, e.g. memory cells and network inputs/outputs. Since these values change throughout the protocol execution, macros are applied to a timestamp term that indicates the time at which the value is read. We assume a set $\mathscr{S}$ of indexed state macro symbols representing memory cells. An indexed state macro symbol is applied to the specified number of indices and also to a single timestamp. In addition to state macros, our logic refers to some predefined macros. The macros $\mathsf{input}@T$ and $\mathsf{output}@T$ are used to refer to the input and output messages of the action executed at time point T. The macros $\mathsf{cond}@T$ and $\mathsf{exec}@T$ respectively encode the condition of the action T, and the conjunction of all the conditions to reach T. The macro $\mathsf{frame}@T$ is used to represent all the messages known by the attacker at time point T.

Given a meta-logic signature $\Sigma = (\mathscr{F}, \mathscr{N}, \mathscr{A}, \mathscr{S})$ and sets of variables $\mathscr{T}$, $\mathscr{X}$, and $\mathscr{I}$ we give in Figure 1 the syntax of meta-terms of sorts timestamp (noted T) and message (noted t), as well as the syntax of local meta-formulas (noted ϕ). Meta-terms of sort index are restricted to variables $i \in \mathscr{I}$. We denote by $\mathsf{fv}(\bullet)$ the free variables (of any sort) of a meta-term or meta-formula.

Example 4. *Going back to the OSK protocol introduced in Section 2, we will typically consider two function symbols* H *and* G *of message arity* 2 *(and index arity* 0*) to represent keyed hash functions. We will also consider the names* $\mathsf{k}, \mathsf{k}' \in \mathscr{N}$ *of index arity* 1*: intuitively,* $\mathsf{k}[i]$ *and* $\mathsf{k}'[i]$ *are the keys used by tag i. We take a single state macro state symbol* s *and another name symbol* n*, both of index arity* 1*: intuitively,* $s[i]@T$ *is the content of tag i's memory cell at time* T*; it is initialized with* $\mathsf{n}[i]$*. Regarding action symbols, this protocol is reduced to a single action symbol* a *of index arity* 2*:* $\mathsf{a}[i,j]$ *is the action performed by tag i for its* j^{th} *session.*

Protocols as sets of actions. Our meta-logic formulas are interpreted as base-logic formulas and their interpretation depends on the protocol under study. We model a protocol as a finite set of actions, each

action representing a basic step of the protocol where the attacker provides an input, a condition is checked, some updates are performed, and finally an output is emitted.

Definition 1. *An* action $\mathsf{a}[i_1,\ldots,i_k].(\phi,o,\{s[j_1,\ldots,j_p]\leftarrow u^s\}_{s\in\mathscr{S}})$ *is formed from an action symbol* a *of index arity* k*, distinct index variables* $i_1,\ldots,i_k$*, a meta-logic formula* ϕ*, a meta-logic term* o *of sort message, and for each state macro symbol* $s\in\mathscr{S}$ *of index arity* p*, and distinct index variables* $j_1,\ldots,j_p$ *(disjoint from* $i_1,\ldots,i_k$*), a meta-logic term* u^s *of sort message.*

Moreover, we assume that $\mathsf{fv}(\phi,o)\subseteq\{i_1,\ldots,i_k\}$ *and* $\mathsf{fv}(u^s)\subseteq\{i_1,\ldots,i_k,j_1,\ldots,j_p\}$*. The formula* ϕ *is called the* condition *of the action,* o *its* output *and* $\{s[j_1,\ldots,j_p]\leftarrow u^s\}_{s\in\mathscr{S}}$ *its state update terms.*

An action $\mathsf{a}[i_1,\ldots,i_k].(\phi,o,\{s[j_1,\ldots,j_p]\leftarrow u^s\}_{s\in\mathscr{S}})$ models that o will be emitted, after having updated each memory cell $s[j_1,\ldots,j_p]$ with the value u^s, 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.

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

Definition 2. *Given a finite set* $\mathscr{A}$ *of action symbols, a protocol* $\mathscr{P}=(\mathscr{P}_{\mathscr{A}},\{s[\vec{j}]\leftarrow u_0^s\}_{s\in\mathscr{S}},<)$ *over* $\mathscr{A}$ *is a finite set* $\mathscr{P}_{\mathscr{A}}$ *of actions, one for each action symbol, a finite set* $\{s[\vec{j}]\leftarrow u_0^s\}_{s\in\mathscr{S}}$ *of initial state values, one for each state macro symbol, equipped with a partial order* $<$ *over terms of the form* $\mathsf{a}[\vec{i}]$ *with* $\mathsf{a}\in\mathscr{A}$*. We require that:*

- *for each memory cell* $s[\vec{j}]$*,* u_0^s *is a macro-free meta-logic term of sort message and* $\mathsf{fv}(u_0^s)\subseteq\{\vec{j}\}$*;*
- $\mathsf{a}[i_1,\ldots,i_k]<\mathsf{a}'[j_1,\ldots,j_{k'}]$ *iff* $\mathsf{a}[\sigma(i_1),\ldots,\sigma(i_k)]<\mathsf{a}'[\sigma(j_1),\ldots,\sigma(j_{k'})]$ *for any* $\mathsf{a},\mathsf{a}',\vec{i}$ *and* $\vec{j}$ *and for any bijective variable renaming* $\sigma:\mathscr{I}\to\mathscr{I}$*;*
- *actions only refer to previously executed actions: for every* $\mathsf{a}[\vec{i}].(\phi,o,\{s[\vec{j}]\leftarrow u^s\}_{s\in\mathscr{S}})\in\mathscr{P}_{\mathscr{A}}$*:*
  - *for any* T *of sort timestamp occurring in the condition* ϕ *or in some* u^s *for* $s\in\mathscr{S}$ *is*
    - *(i) either* T *is of the form* $\mathsf{a}'[\vec{j}]$ *with* $\mathsf{a}'[\vec{j}]<\mathsf{a}[\vec{i}]$*,*
    - *(ii) or* T *is of the form* $\mathsf{pred}^n(\mathsf{a}[\vec{i}])$ *with* $n\geq 1$*,*
    - *(iii) or* T *is* $\mathsf{a}[\vec{i}]$ *and appears in an input macro* $\mathsf{input}@\mathsf{a}[\vec{i}]$*,*
  - *for any* T *of sort timestamp occurring in* o*, we have either (i) or (ii) or (iii), or*
    - *(iv)* T *is of the form* $\mathsf{a}[\vec{i}]$ *and appears in a state macro* $s[\vec{l}]@\mathsf{a}[\vec{i}]$*.*

Intuitively, an action can refer to its own input, or to the value of a state at the timestamp corresponding to this action if it is in the output term (i.e. after all updates have been performed), and can otherwise only refer to timestamps corresponding to actions that accrued strictly before it.

Example 5. *Considering a scenario of the OSK protocol involving multiple tags, each tag playing multiple sessions, we have that this protocol is reduced to a unique action parametrized by* i *and* j *to identify the tag and the session. An execution of the OSK protocol, from the point of view of the tag* i *consists of updating its state* $s[i]$ *using* H*, and outputting its content after the application of the keyed hash function* G*. In our syntax, we have that* $\mathscr{P}=(\mathscr{P}_{\mathscr{A}},\{s[i']\leftarrow u_0^s\}_{s\in\mathscr{S}},<)$ *where* $<$ *is the empty dependency relation,* $\mathscr{P}_A$ *contains* $\mathsf{a}[i,j].(\phi,o,\{s[i']\leftarrow u^s\}_{s\in\mathscr{S}})$ *where* $\phi=\top$*,* $o=\mathsf{G}(s[i]@\mathsf{a}[i,j],k'[i])$*,* $u_0^s=\mathsf{n}[i']$*, and*

$$u^s \stackrel{def}{=} \mathsf{if}\ i'=i\ \mathsf{then}\ \mathsf{H}(s[i]@\mathsf{pred}(\mathsf{a}[i,j]),k[i])\ \mathsf{else}\ s[i']@\mathsf{pred}(\mathsf{a}[i,j]).$$

3.2 Semantics

To give a semantics to meta-terms and local meta-formulas, we translate them to terms of the base logic. This translation depends on the number of protocol agents and sessions, and on the interleaving of the protocol actions (i.e., the order in which the adversary interacts with the different protocol agents). This information is going to be encapsulated by the notion of *trace model*, which we are now going to define.

Protocol execution. We can instantiate the indices of an action by values to yield *concrete actions*.

Definition 3. *Given a set $\mathscr{A}$ of action symbols, a* concrete action *is the application of an action symbol $a \in \mathscr{A}$ to k* integers *(k is the index arity of a).*

We lift the partial order of a protocol $\mathscr{P}$ to concrete actions: $a[\sigma(i_1),\dots,\sigma(i_k)] < b[\sigma(j_1),\dots,\sigma(j_l)]$ holds when $a[i_1,\dots,i_k] < b[j_1,\dots,j_l]$ for any $\sigma : \mathscr{I} \to \mathbb{N}$.

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 4. *Given a protocol $\mathscr{P}$, 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, addressed below through the exec macros.

Trace model. We consider meta-logic terms and formulas over $\Sigma = (\mathscr{F}, \mathscr{N}, \mathscr{A}, \mathscr{S})$, and given a finite set D of integers, the associated base logic signature $\Sigma^D = (\mathscr{F}_B, \mathscr{N}_B)$ contains exactly a name symbol $\mathsf{n}_{k_1,\dots,k_p}$ for every $\mathsf{n} \in \mathscr{N}$ of index arity p, and every $k_1,\dots,k_p \in D$; and a function symbol $\mathsf{f}_{k_1,\dots,k_p}$ of arity n for every $\mathsf{f} \in \mathscr{F}$ of index arity p and message arity n, and every $k_1,\dots,k_p \in D$.

In order to interpret meta-terms and meta-formulas, we introduce the notion of trace model. The idea is that for each interleaving of the actions of the protocol under study, we can define a structure that will allow us to give a meaning to the macros.

Definition 5. *A trace model $\mathbb{T}$ (of a protocol $\mathscr{P}$) is a tuple $(\mathscr{D}_\mathscr{I}, \mathscr{D}_\mathscr{T}, <_\mathscr{T}, \sigma_\mathscr{I}, \sigma_\mathscr{T})$ such that:*

- *$\mathscr{D}_\mathscr{I} \subseteq \mathbb{N}$ is a finite index domain;*
- *$\mathscr{D}_\mathscr{T}$ contains two special symbols* init*, and* undef*, as well as a subset of*

$$\mathscr{D}_a \stackrel{def}{=} \{a[k_1,\dots,k_n] \mid a \in \mathscr{A}, k_1,\dots,k_n \in \mathscr{D}_\mathscr{I}\};$$

- *$<_\mathscr{T}$ is a total ordering on $\mathscr{D}_\mathscr{T} \setminus \{\mathsf{undef}\}$ such that* init *is minimal, and such that the sequence of elements of $\mathscr{D}_\mathscr{T} \setminus \{\mathsf{undef}\}$ ordered by $<_\mathscr{T}$ is an interleaving of $\mathscr{P}$;*
- *$\sigma_\mathscr{I} : \mathscr{I} \to D_\mathscr{I}$ and $\sigma_\mathscr{T} : \mathscr{T} \to D_\mathscr{T}$ are mappings that interpret index and timestamp variables as elements of their respective domains.*

Given a trace model, we define a predecessor function $\mathsf{pred}_\mathscr{T} : \mathscr{D}_\mathscr{T} \to \mathscr{D}_\mathscr{T}$ which maps undef to itself, init to undef, and all other elements v to the largest element v' such that $v' <_\mathscr{T} v$. Moreover, we denote by $\leq_\mathscr{T}$ the reflexive closure of $<_\mathscr{T}$ on $\mathscr{D}_\mathscr{T} \setminus \{\mathsf{undef}\}$.

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

Translation. We can now give our translation for meta-terms and meta-formulas. This translation is similar to the one we introduced in [1], but adapted to our notion of trace model in which all actions do not necessarily happen. For sake of clarity, we recall its general principle while highlighting the main differences with the translation given in [1].

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

Figure 2: Interpretation of macros, where $\mathsf{a}[\vec{i}].(\phi, o, \{s[\vec{j}] \leftarrow u^s\}_{s\in\mathscr{S}})$ is an action of $\mathscr{P}$.

The translation $(\bullet)^{\mathbb{T}}_{\mathscr{P}}$ is parametrized by a protocol $\mathscr{P}$ and a trace model $\mathbb{T}$ of $\mathscr{P}$. First, as expected, we have that $(\tau)^{\mathbb{T}}_{\mathscr{P}} = \sigma_{\mathscr{T}}(\tau)$, $(\mathsf{init})^{\mathbb{T}}_{\mathscr{P}} = \mathsf{init}$, and $(\mathsf{pred}(T))^{\mathbb{T}}_{\mathscr{P}} = \mathsf{pred}_{\mathscr{T}}((T)^{\mathbb{T}}_{\mathscr{P}})$. Then, each meta-logic construct is translated using its counterpart in the base logic when it is available. A name $\mathsf{n}[i_1,\ldots,i_p]$ is translated into $\mathsf{n}_{\sigma_{\mathscr{I}}(i_1),\ldots,\sigma_{\mathscr{I}}(i_p)}$. For some constructions, it is a bit more complicated. For instance, the lookup construct is not available in the base logic but it can be translated relying on nested conditionals. The interpretation of meta-formulas is quite straightforward using finite boolean expressions to translate quantifications over index and timestamp variables.

Then, regarding atomic meta-formula involving timestamps, we have that:

- $(T = T')^{\mathbb{T}}_{\mathscr{P}} = \mathsf{true}$ iff $(T)^{\mathbb{T}}_{\mathscr{P}} = (T')^{\mathbb{T}}_{\mathscr{P}}$;
- $(T < T')^{\mathbb{T}}_{\mathscr{P}} = \mathsf{true}$ iff $(T)^{\mathbb{T}}_{\mathscr{P}} <_{\mathscr{T}} (T')^{\mathbb{T}}_{\mathscr{P}}$, and similarly for $\leq$;
- $(\mathsf{happens}(T))^{\mathbb{T}}_{\mathscr{P}} = \mathsf{true}$ iff $(T)^{\mathbb{T}}_{\mathscr{P}} \in \mathscr{D}_{\mathscr{T}} \setminus \{\mathsf{undef}\}$.

Lastly, we have to explain the interpretation of our macros. We recall in Figure 2 the terms m_{init} and $\{m_{\mathsf{a}[\vec{i}]} \mid \mathsf{a}[\vec{i}] \in \mathscr{P}_{\mathscr{A}}\}$ for the macro symbols cond, output, input, frame, and exec. Then, we have that:

$$(m@T)^{\mathbb{T}}_{\mathscr{P}} = \begin{cases} \mathsf{empty} & \text{if } (T)^{\mathbb{T}}_{\mathscr{P}} = \mathsf{undef}\\ (m_{\mathsf{init}})^{\mathbb{T}}_{\mathscr{P}} & \text{if } (T)^{\mathbb{T}}_{\mathscr{P}} = \mathsf{init}\\ (m_{\mathsf{a}[\vec{i}]})^{\mathbb{T}\{\vec{i}\mapsto\vec{k}\}}_{\mathscr{P}} & \text{if } (T)^{\mathbb{T}}_{\mathscr{P}} = \mathsf{a}[\vec{k}] \in \mathscr{D}_{\mathscr{T}} \setminus \{\mathsf{init},\mathsf{undef}\} \text{ and } \mathsf{a}[\vec{i}] \in \mathscr{P}_{\mathscr{A}} \end{cases}$$

Roughly, an output macro is replaced by the meta-term as specified by the protocol, and is then interpreted in the trace model $\mathbb{T}$ to get a term of the base logic. The cond macro has a similar treatment and produces a base formula corresponding to the conditional of the action. The macro exec is translated as the conjunction of all past conditions. The translation of the frame macro 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, obtains the output. Finally, in order to model the fact that the attacker controls the network, the input macro is interpreted using the attacker symbol att, to which we pass the current frame.

The translation of state macros in $\mathscr{P} = (\mathscr{P}_{\mathscr{A}}, \mathscr{P}_{\mathscr{U}}, <)$ is defined as follows:

$$(s[\vec{l}]@T)^{\mathbb{T}}_{\mathscr{P}} \stackrel{\text{def}}{=} \begin{cases} \mathsf{empty} & \text{if } (T)^{\mathbb{T}}_{\mathscr{P}} = \mathsf{undef}\\ (u^s_0)^{\mathbb{T}\{\vec{j}\mapsto\vec{k}_0\}}_{\mathscr{P}} & \text{if } (T)^{\mathbb{T}}_{\mathscr{P}} = \mathsf{init}, (\vec{l})^{\mathbb{T}}_{\mathscr{P}} = \vec{k}_0 \text{ and } (s[\vec{j}] \leftarrow u^s_0) \in \mathscr{P}_{\mathscr{U}}\\ (u^s)^{\mathbb{T}\{\vec{i}\mapsto\vec{k},\vec{j}\mapsto\vec{k}_0\}}_{\mathscr{P}} & \text{if } (T)^{\mathbb{T}}_{\mathscr{P}} = \mathsf{a}[\vec{k}] \in \mathscr{D}_{\mathscr{T}} \setminus \{\mathsf{init},\mathsf{undef}\},\ (\vec{l})^{\mathbb{T}}_{\mathscr{P}} = \vec{k}_0\\ & \quad \text{where } \mathsf{a}[\vec{i}].(\phi, o, \mathscr{U}) \in \mathscr{P}_{\mathscr{A}} \text{ and } (s[\vec{j}] \leftarrow u^s) \in \mathscr{U} \end{cases}$$

A local meta-logic formula ϕ is said to be valid w.r.t. a protocol $\mathscr{P}$ when, for any trace model $\mathbb{T}$, the base logic formula $(\phi)^{\mathbb{T}}_{\mathscr{P}} \sim \mathsf{true}$ is valid (i.e., it is satisfied in all computational models).

Example 6. *The meta-formula* $\neg(\mathsf{happens}(\tau_1)) \Rightarrow \neg(\mathsf{happens}(\tau_2)) \Rightarrow \tau_1 = \tau_2$ *is valid. The same goes for* $\tau_1 < \tau_2 \Rightarrow \mathsf{happens}(\tau_1) \wedge \mathsf{happens}(\tau_2)$*, and similarly for* $\leq$*, but this does not hold for* $=$*.*

Example 7. *We can express that an adversary cannot forge future outputs of OSK tags as follows in the meta-logic:* $\forall i, j, \tau.\ \ \tau < a[i,j] \Rightarrow \mathsf{input}@\tau \neq \mathsf{output}@a[i,j]$*. The validity of this formula corresponds to the validity of its translation for all trace models, i.e. for any execution trace of any finite set of tags. The formula is indeed valid, but only if one restricts computational models to those where* G *and* H *are interpreted as PRF hash functions.*

In the notion of trace model we introduced in [1], the set $\mathscr{D}_{\mathscr{T}}$ contains init and *all* concrete actions of the set $\mathscr{D}_a$. In the new definition, $\mathscr{D}_{\mathscr{T}}$ contains only a subset of all possible concrete actions, which correspond to the actions that happen. This change allows to model conflicts between actions: for instance, the axiom $\forall \vec{i}, \vec{j}.\ \neg\big(\mathsf{happens}(\mathsf{a}_1[\vec{i}]) \wedge \mathsf{happens}(\mathsf{a}_2[\vec{j}])\big)$ would rule out traces where the two actions are scheduled. It also becomes possible to model phases using axioms of the following form:

$$\forall \vec{i}, \vec{j}.\ \mathsf{happens}(\mathsf{a}_1[\vec{i}]) \Rightarrow \mathsf{happens}(\mathsf{a}_2[\vec{j}]) \Rightarrow \mathsf{a}_1[\vec{i}] < \mathsf{a}_2[\vec{j}]$$

With the new trace models, this excludes traces where a_2 actions are scheduled before a_1 actions. With the old trace model, we could only force *all* a_1 actions to happen before a_2 actions, which is not realistic — some of these actions might in fact be mutually exclusive.

3.3 Global meta-logic formulas

Local meta-logic formulas are interpreted to boolean terms. We now introduce *global meta-formulas* that will be interpreted as base-logic formulas, where we write α for a variable of sort index, timestamp or message:

$$F ::= 0 \mid [\phi]_{\mathscr{P}} \mid [\vec{t} \sim \vec{t'}]_{\mathscr{P},\mathscr{P}'} \mid F \to F' \mid \Pi\alpha.F$$

To avoid the confusion with local meta-formulas we use other symbols for universal quantification and implication, and we also make explicit the protocols under study since different parts of the formula may refer to different protocols.

Atoms are either reachability formulas, relative to some protocol, or equivalence properties, relative to two protocols. All protocols used in a global meta-formula must be compatible, i.e. they are based on the same set $\mathscr{A}$ of action names, and have the same partial order. Note that, unlike in local meta-formulas, we can quantify over messages in global meta-formulas. As usual, other propositional connectives and quantifiers can be expressed using the ones given above.

Example 8. *For instance, we can express the fact that the length of* $\langle m_1, m_2 \rangle$ *(concatenation of* m_1 *and* m_2*) is equal to the sum of the length of each component as follows:*

$$\Pi x_1.\Pi x_2.[\mathsf{len}(\langle x_1, x_2 \rangle) = \mathsf{plus}(\mathsf{len}(x_1), \mathsf{len}(x_2))]_{\mathscr{P}}$$

This global meta-formula depends neither on the protocol under study nor on its trace model as no action is present in the formula. We can also rely on these formulas to express restrictions on traces that are specific to the protocol: we may impose that some actions occur before some others, or the sequentiality between actions to model the fact that a tag is involved in at most one session at a time (see Section 5).

A global meta-formula is said to be valid when, for any trace model, its meta-interpretation as a base-logic formula is valid. This meta-interpretation is defined as follows:

$$\begin{array}{rcl@{\qquad}rcl} (0)^{\mathbb{T}} = \bot & & & & & \\ & ([\phi]_{\mathscr{P}})^{\mathbb{T}} & = & (\phi)^{\mathbb{T}}_{\mathscr{P}} \sim \mathsf{true} & (\Pi i.F)^{\mathbb{T}} & = & \wedge_{v \in \mathscr{D}_{\mathscr{I}}} (F)^{\mathbb{T}\{i \mapsto v\}} \\ & ([\vec{t} \sim \vec{t'}]_{\mathscr{P},\mathscr{P}'})^{\mathbb{T}} & = & (\vec{t})^{\mathbb{T}}_{\mathscr{P}} \sim (\vec{t'})^{\mathbb{T}}_{\mathscr{P}'} & (\Pi\tau.F)^{\mathbb{T}} & = & \wedge_{v \in \mathscr{D}_{\mathscr{T}}} (F)^{\mathbb{T}\{\tau \mapsto v\}} \\ & (F \to F')^{\mathbb{T}} & = & (F)^{\mathbb{T}} \Rightarrow (F')^{\mathbb{T}} & (\Pi x.F)^{\mathbb{T}} & = & \forall x.(F)^{\mathbb{T}} \end{array}$$

Definition 6. *Two protocols $\mathscr{P}_1$ and $\mathscr{P}_2$ are* observationally equivalent *when they are compatible and the following global meta-logic formula is valid:*

$$\Pi\tau.\ \big([\mathsf{happens}(\tau)]_{\mathscr{P}_1} \wedge [\mathsf{happens}(\tau)]_{\mathscr{P}_2}\big) \to [\mathsf{frame@}\tau \sim \mathsf{frame@}\tau]_{\mathscr{P}_1,\mathscr{P}_2}$$

Note that $[\mathsf{happens}(\tau)]_{\mathscr{P}_1}$ and $[\mathsf{happens}(\tau)]_{\mathscr{P}_2}$ are equivalent as the meta-formula $\mathsf{happens}(\tau)$ only talks about the trace model which is common to compatible protocols.

4 Proof systems

We introduce here some notions of sequents that generalize the ones given in [1]. This generalization enables a fully formal account of the use of axioms, trace restrictions, and lemmas. It also enables richer reasoning, as illustrated in the next section.

We have two kinds of sequents, general and reachability sequents, which are respectively of the form

$$\Sigma; \Theta \vdash F \qquad \text{and} \qquad \Sigma; \Theta : \Gamma \vdash_{\mathscr{P}} \phi$$

where Σ is a sequence of variables (of any sort), Θ is a set of global meta-formulas, Γ is a set of local meta-formulas, ϕ is a local meta-formula and F is a global meta-formula. We require that sequents are closed, i.e. Σ binds all variables occurring in the rest of the sequent.

We shall define the translation of sequents to global meta-formulas, which indirectly defines their semantics. When $\Theta = \{F_1, \ldots, F_n\}$ we write $\Theta \to F$ for $F_1 \to \ldots \to F_n \to F$. We define similarly the notation $\Gamma \Rightarrow \phi$. Finally, when $\Sigma = \{\alpha_1, \ldots, \alpha_n\}$, we write $\Pi\Sigma.F$ for $\Pi\alpha_1 \ldots \Pi\alpha_n.F$. The translation of our two kinds of sequents is then defined as follows:

$$\Sigma; \Theta \vdash F \rightsquigarrow \Pi\Sigma.(\Theta \to F) \qquad \text{and} \qquad \Sigma; \Theta : \Gamma \vdash_{\mathscr{P}} \phi \rightsquigarrow \Pi\Sigma.(\Theta \to [\Gamma \Rightarrow \phi]_{\mathscr{P}})$$

A sequent is valid when its translation as a global meta-formula is valid.

Example 9. *To understand the distinction between Θ and Γ hypotheses in reachability sequents, note that the validity of $\Sigma; : \phi \vdash_{\mathscr{P}} \psi$ implies that of $\Sigma; [\phi]_{\mathscr{P}} :\vdash_{\mathscr{P}} \psi$, but the converse does not generally hold. In the former case we state that the implication $\phi \Rightarrow \psi$ is true with overwhelming probability; in the latter, we state that if ϕ is true with overwhelming probability then so is ψ.*

Conveniently, inferences that are valid for classical first-order logic can be lifted to our reachability sequents. General sequents are also equipped with a proof system which amounts to classical natural deduction (some example of reachability and general rules are in Appendix A). General sequents subsume the equivalence sequents of [1]: when $\vec{u}$ and $\vec{v}$ are sequences of meta-terms of the same length, $\Sigma; \Theta \vdash_{\mathscr{P},\mathscr{P}'} \vec{u} \sim \vec{v}$ is a notation for $\Sigma; \Theta \vdash [\vec{u} \sim \vec{v}]_{\mathscr{P},\mathscr{P}'}$. We show in the upper-part of Figure 3 some rules that are specific to such sequents.

Our proof system allows two forms of induction over timestamps: the first is specific to reachability sequents, and ignores Θ, and the second is at the global meta-level, and generalizes the induction rule given in [1] for equivalence sequents.

$$\textsc{Local-Induction}\ \frac{\Sigma; \Theta : \Gamma \vdash_{\mathscr{P}} \forall\tau.\ (\forall\tau'.\ \tau' < \tau \Rightarrow \phi) \Rightarrow \phi}{\Sigma; \Theta : \Gamma \vdash_{\mathscr{P}} \forall\tau.\phi} \qquad \textsc{Global-Induction}\ \frac{\Sigma; \Theta \vdash \Pi\tau.\ (\Pi\tau'.\ \tau' < \tau \to F) \to F}{\Sigma; \Theta \vdash \Pi\tau.F}$$

The lower part of Figure 3 presents some rules which articulate the relationship between our various sequents. EQUIV-TERM and EQUIV-FORM are immediate generalization of the identically named

Inference rules for equivalence sequents.

PERMUT
$$\frac{\Sigma;\Theta\vdash_{\mathscr{P},\mathscr{P}'} u_{\sigma(1)},\dots,u_{\sigma(n)}\sim v_{\sigma(1)},\dots,v_{\sigma(n)}}{\Sigma;\Theta\vdash_{\mathscr{P},\mathscr{P}'} u_1,\dots,u_n\sim v_1,\dots,v_n}$$

FA
$$\frac{\Sigma;\Theta\vdash_{\mathscr{P},\mathscr{P}'} \vec{u},t_1,\dots,t_n\sim\vec{v},t'_1,\dots,t'_n}{\Sigma;\Theta\vdash_{\mathscr{P},\mathscr{P}'} \vec{u},f(t_1,\dots,t_n)\sim\vec{v},f(t'_1,\dots,t'_n)}$$

Inference rules involving mixed kinds of sequents.

EQUIV-TERM
$$\frac{\Sigma;\Theta:\vdash_{\mathscr{P}} t=t' \qquad \Sigma;\Theta\vdash_{\mathscr{P},\mathscr{P}'}\vec{C}[t']\sim\vec{v}}{\Sigma;\Theta\vdash_{\mathscr{P},\mathscr{P}'}\vec{C}[t]\sim\vec{v}}$$

EQUIV-FORM
$$\frac{\Sigma;\Theta:\vdash_{\mathscr{P}} \phi\Leftrightarrow\psi \qquad \Sigma;\Theta\vdash_{\mathscr{P},\mathscr{P}'}\vec{C}[\psi]\sim\vec{v}}{\Sigma;\Theta\vdash_{\mathscr{P},\mathscr{P}'}\vec{C}[\phi]\sim\vec{v}}$$

REACH-EQUIV
$$\frac{\Sigma;\Theta\vdash_{\mathscr{P},\mathscr{P}'}\vec{u}\sim\vec{v} \qquad \Sigma;\Theta:\Gamma[\vec{v}]\vdash_{\mathscr{P}'}\phi[\vec{v}]}{\Sigma;\Theta:\Gamma[\vec{u}]\vdash_{\mathscr{P}}\phi[\vec{u}]}$$

GLOBAL-LOCAL
$$\frac{\Sigma;\Theta\vdash[\phi]_{\mathscr{P}}}{\Sigma;\Theta:\vdash_{\mathscr{P}}\phi}$$

LOCAL-GLOBAL
$$\frac{\Sigma;\Theta:\vdash_{\mathscr{P}}\phi}{\Sigma;\Theta\vdash[\phi]_{\mathscr{P}}}$$

In PERMUT, σ is a bijection on $[1;n]$. In REACH-EQUIV, we assume that Γ and ϕ are contexts that do not contain any name nor macros: in other words, any such subterms must be treated as part of $\vec{u}$.

Figure 3: Some inference rules for general sequents.

rules in [1]: we add a new context Θ, which allows to carry hypotheses from the equivalence sequent in conclusion to the reachability sequent in first premise. Conversely, REACH-EQUIV shows how an equivalence judgment can be used to help derive reachability judgments. Finally, GLOBAL-LOCAL and LOCAL-GLOBAL allow to move from one kind of sequent to the other when Γ is empty.

5 Application

Consider two keyed hash functions H and G, assuming PRF on both. We will use H with a fixed key k, and G with k'. Consider the OSK protocol introduced in Example 1 that has a mutable state s, initially holding a name n, and can perform the following action any number of times: update s with $\mathsf{H}(s,\mathsf{k})$, output $\mathsf{G}(s,\mathsf{k}')$. In Example 5, we gave the definition of an action $\mathsf{a}[i,j]$ modelling a scenario of the OSK protocol involving multiple tags. In this section, we will only consider a scenario involving a single tag, which can still play multiple sessions. In addition, we provide the attacker with the ability to compute hashes with their respective keys, which corresponds to the random oracle model. Let $\mathscr{P}$ be the protocol corresponding to this process: it features actions $\mathsf{a}[i]$ and $\mathsf{o}[j]$, and of course the initial action. The output of the action $\mathsf{o}[j]$ is defined by the pair $\langle\mathsf{H}(\mathsf{input@o}[j],\mathsf{k}),\mathsf{G}(\mathsf{input@o}[j],\mathsf{k}')\rangle$.

In this protocol, the outputs $\mathsf{G}(s@\mathsf{a}[i],\mathsf{k}')$ are strongly secret, i.e., indistinguishable from random values. Intuitively, this holds even though the attacker can compute hashes (using the oracle) because it cannot guess the values of s, and hence cannot predict the output of action $\mathsf{a}[i]$. Moreover, there is a negligible probability that the successive values of s repeat. In this analysis, it only matters that the successive values of s are non-repeating and weakly secret, for which the collision resistance of H is enough.

Using the PRF assumption of H, we will actually prove the strong secrecy of the states $s@\mathsf{a}[i]$, from which the strong secrecy of the outputs $\mathsf{G}(s@\mathsf{a}[i],\mathsf{k}')$ will follow. These developments use some rules

of our proof system that were not explicitly detailed in Section 4. We will explain their behaviour on examples, and refer the reader to [1] for detailed definitions. The rule PRF reflects the PRF cryptographic assumption for hash functions. Informally, it states that a hashed message can be replaced by a fresh name if this message has not been previously hashed in the past. Another rule, FRESH, allows to remove a name on both sides of an equivalence sequent if it does not occur anywhere else (i.e. the name is fresh).

5.1 Secrecy of states

For simplicity, we will assume that the attacker never repeats a query to the oracle. We thus seek to prove the following global meta-formula, where m is a fresh name and ϕ_{restr} is the local meta-formula $\forall j, j'.\ j \neq j' \Rightarrow \mathsf{input@o}[j] \neq \mathsf{input@o}[j']$:

$$[\phi_{\text{restr}}]_{\mathscr{P}} \to \Pi\tau.\ [\mathsf{happens}(\tau)]_{\mathscr{P}} \to \Pi i.\ [\mathsf{happens}(\mathsf{a}[i])]_{\mathscr{P}} \to [s@\mathsf{a}[i], \mathsf{frame@}\tau \sim \mathsf{m}, \mathsf{frame@}\tau]_{\mathscr{P},\mathscr{P}}$$

This property says that an attacker (interacting with the tags and oracle) cannot distinguish $s@\mathsf{a}[i]$ from a random sampling m. This is stronger than simply saying that this value is unknown to the attacker.

We first state a lemma that will be useful later on. This lemma says that values stored in the state s never repeat, expressed with the following local meta-formula:

$$\forall \tau, i.\ \tau < \mathsf{a}[i] \Rightarrow s@\tau \neq s@\mathsf{a}[i]$$

This lemma can be proved by induction and relies on the collision resistance of H.

We prove the secrecy of states using GLOBAL-INDUCTION on τ, followed by a case analysis.

Case where $\tau = \mathsf{init}$. We essentially have to prove the following sequent, since $\mathsf{frame@}\tau = \mathsf{empty}$:

$$i; \Theta \vdash [s@\mathsf{a}[i] \sim \mathsf{m}]_{\mathscr{P},\mathscr{P}}$$

where Θ contains $[\phi_{\text{restr}}]_{\mathscr{P}}$, $[\mathsf{happens}(\mathsf{init})]_{\mathscr{P}}$, $[\mathsf{happens}(\mathsf{a}[i])]_{\mathscr{P}}$ and the induction hypothesis, which will be unusable in this case since init is the base case. We can use PRF on $s@\mathsf{a}[i] = \mathsf{H}(s@\mathsf{pred}(\mathsf{a}[i]), \mathsf{k}[i])$ to replace the state by m. The rule generates the following condition, expressing that $s@\mathsf{pred}(\mathsf{a}[i])$ must not have been previously hashed[2]:

$$i; \Theta :\vdash_{\mathscr{P}} \forall i'.\ \mathsf{a}[i'] \leq \mathsf{pred}(\mathsf{a}[i]) \Rightarrow s@\mathsf{pred}(\mathsf{a}[i']) \neq s@\mathsf{pred}(\mathsf{a}[i])$$

This follows from the lemma on non-repeating states, since $s@\mathsf{pred}(\mathsf{a}[i])$ must be equal to some $s@\mathsf{a}[i'']$ for which we still have $\mathsf{pred}(\mathsf{a}[i']) < \mathsf{a}[i'']$.

Case where $\tau = \mathsf{o}[j]$. We use the FA rule (given in Figure 3) to split the pair, and are left with the following sequent to prove:

$$j, i; \Theta \vdash [s@\mathsf{a}[i], \mathsf{frame@pred}(\mathsf{o}[j]), u_1, u_2 \sim \mathsf{m}, \mathsf{frame@pred}(\mathsf{o}[j]), u_1, u_2]_{\mathscr{P},\mathscr{P}}$$

where $u_1 = \mathsf{H}(\mathsf{input@o}[j], \mathsf{k})$, $u_2 = \mathsf{G}(\mathsf{input@o}[j], \mathsf{k}')$, and Θ contains our induction hypothesis, ϕ_{restr} as well as $\mathsf{happens}(\mathsf{o}[j])$ and $\mathsf{happens}(\mathsf{a}[i])$. We handle each hashed message using PRF, as follows.

[2] This temporal intuition is formally reflected by considering all possible subterms of the form $\mathsf{H}(\_, \mathsf{k}_i)$ in the meta-interpretations of $s@\mathsf{pred}(\mathsf{a}[i])$. We obtain an over-approximation of this set of terms by considering all possible outputs, updates and conditions of actions scheduled before $\mathsf{pred}(\mathsf{a}[i])$.

1. We use PRF on the last item of the sequences, i.e. u_2. The generated condition is [3]:

$$\begin{array}{rl} j,i;\Theta :\vdash_{\mathscr{P}} & \forall j'.\ \mathsf{o}[j'] < \mathsf{o}[j] \Rightarrow \mathsf{input@o}[j'] \neq \mathsf{input@o}[j] \\ \wedge & \forall i'.\ \mathsf{a}[i'] < \mathsf{o}[j] \Rightarrow s@\mathsf{a}[i'] \neq \mathsf{input@o}[j] \end{array}$$

   The first conjunct is an easy consequence of $[\phi_{\text{rest}}]_{\mathscr{P}}$. For the second conjunct we use the induction hypothesis with $\tau := \mathsf{pred}(\mathsf{o}[j])$ and $i := i'$. This gives us

$$j,i,i';\Theta \vdash \left[s@\mathsf{a}[i'], \mathsf{frame@pred}(\mathsf{o}[j]) \sim \mathsf{m}, \mathsf{frame@pred}(\mathsf{o}[j])\right]_{\mathscr{P},\mathscr{P}}$$

   which, together with the fact that $\mathsf{input@o}[j] = \mathsf{att}(\mathsf{frame@pred}(\mathsf{o}[j]))$, allows us to replace $s@\mathsf{a}[i']$ by m in our goal using our new rule REACH-EQUIV (see Figure 3). It remains to prove $j,i,i';\Theta : \ldots \vdash_{\mathscr{P}} \mathsf{m} \neq \mathsf{input@o}[j]$ which is easily done using rule Fresh: indeed, m has nothing to do with protocol $\mathscr{P}$ (formally, it cannot occur in any meta-interpretation of $\mathsf{input@o}[j]$).

2. We then use PRF on u_1. The generated condition will feature two conjuncts: one for a previous oracle query, and one corresponding to an update which may occur before $\mathsf{a}[i]$ or $\mathsf{pred}(\mathsf{o}[j])$.

$$\begin{array}{rl} j,i;\Theta :\vdash & \left(\forall j'.\ \mathsf{o}[j'] \leq \mathsf{pred}(\mathsf{o}[j]) \Rightarrow \mathsf{input@o}[j'] \neq \mathsf{input@o}[j]\right) \\ & \wedge \left(\forall i'.\ \mathsf{a}[i'] \leq \mathsf{pred}(\mathsf{o}[j]) \vee \mathsf{a}[i'] \leq \mathsf{a}[i] \Rightarrow s@\mathsf{pred}(\mathsf{a}[i']) \neq \mathsf{input@o}[j]\right) \end{array}$$

   As before, we prove the first conjunct using ϕ_{restr}. To prove the second conjunct, we first observe that $s@\mathsf{pred}(\mathsf{a}[i'])$ is either equal to $s@\mathsf{init}$ or to some $s@\mathsf{a}[i'']$. We omit the proof for the first case[4]. In the second case, the induction hypothesis with $\tau := \mathsf{pred}(\mathsf{o}[j])$ and $i := i''$ gives us

$$j,i,i',i'';\Theta \vdash \left[s@\mathsf{pred}(\mathsf{a}[i']), \mathsf{frame@pred}(\mathsf{o}[j]) \sim \mathsf{m}, \mathsf{frame@pred}(\mathsf{o}[j])\right]_{\mathscr{P},\mathscr{P}}$$

   which allows us to replace $s@\mathsf{pred}(\mathsf{a}[i'])$ by m in our goal using our new rule REACH-EQUIV. It remains to prove $j,i,i',i'';\Theta : \ldots \vdash_{\mathscr{P}} \mathsf{m} \neq \mathsf{input@o}[j]$ which we derive using rule FRESH.

Now that the two hashed messages are removed from our goal, we conclude by induction hypothesis with $\tau := \mathsf{pred}(\mathsf{o}[j])$ and $i := i$.

Case where $\tau = \mathsf{a}[i']$. We need to prove

$$i',i;\Theta \vdash [s@\mathsf{a}[i], \mathsf{frame@pred}(\mathsf{a}[i']), v_1 \sim \mathsf{m}, \mathsf{frame@pred}(\mathsf{a}[i']), v_1]_{\mathscr{P},\mathscr{P}}$$

where $v_1 = \mathsf{G}(s@\mathsf{a}[i'], \mathsf{k}')$. In order to conclude by induction hypothesis, we get rid of v_1 using the PRF rule. This time, the generated condition is:

$$\begin{array}{rl} i',i;\Theta :\vdash_{\mathscr{P}} & \forall j'.\ \mathsf{o}[j'] < \mathsf{a}[i'] \Rightarrow \mathsf{input@o}[j'] \neq s@\mathsf{a}[i'] \\ \wedge & \forall i'.\ \mathsf{a}[i''] < \mathsf{a}[i'] \Rightarrow s@\mathsf{a}[i''] \neq s@\mathsf{a}[i'] \end{array}$$

The second conjunct is a consequence of the non-repeating states lemma. For the first one, we apply the induction hypothesis to obtain $\left[s@\mathsf{a}[i'], \mathsf{frame@pred}(\mathsf{a}[i']) \sim \mathsf{m}, \mathsf{frame@pred}(\mathsf{a}[i'])\right]_{\mathscr{P},\mathscr{P}}$. Since $\mathsf{pred}(\mathsf{o}[j']) \leq \mathsf{pred}(\mathsf{a}[i'])$, this implies $\left[s@\mathsf{a}[i'], \mathsf{frame@pred}(\mathsf{o}[j']) \sim \mathsf{m}, \mathsf{frame@pred}(\mathsf{o}[j'])\right]_{\mathscr{P},\mathscr{P}}$, which justifies the replacement of $s@\mathsf{a}[i']$ by m in our goal using rule REACH-EQUIV. We conclude using rule FRESH.

[3] We assume here a simpler condition than what the current tool would generate, exploiting the fact that no hashes using G may found in meta-interpretations of $s@\mathsf{a}[i]$.

[4] This proof is not entirely trivial, and would be probably be best avoided by slightly generalizing the strong secrecy statement. We believe, however, that this issue is orthogonal to the main point illustrated by our application.

5.2 Strong secrecy of outputs

We now establish the strong secrecy of $\mathsf{a}[i]$ outputs. Let $\mathscr{P}'$ be a modified version of $\mathscr{P}$ where the output of $\mathsf{a}[i]$ is a fresh name $\mathsf{m}[i]$, and the mutable cell s is never updated. Our strong secrecy property is expressed as the observational equivalence of $\mathscr{P}$ and $\mathscr{P}'$:

$$[\phi_{\text{restr}}]_{\mathscr{P}} \to \Pi\tau.\ [\mathsf{happens}(\tau)]_{\mathscr{P}} \to \left[\mathsf{frame}@\tau \sim \mathsf{frame}@\tau\right]_{\mathscr{P},\mathscr{P}'}$$

We prove this using the GLOBAL-INDUCTION rule. The init case is trivial.

Case where $\tau = \mathsf{o}[j]$. We use the FA rule to split the pair, and we are left with the following sequent to prove:

$$j; \Theta \vdash \left[\mathsf{frame}@\mathsf{pred}(\mathsf{o}[j]), u_1, u_2 \sim \mathsf{frame}@\mathsf{pred}(\mathsf{o}[j]), u_1, u_2\right]_{\mathscr{P},\mathscr{P}'}$$

where $u_1 = \mathsf{H}(\mathsf{input}@\mathsf{o}[j], \mathsf{k})$, $u_2 = \mathsf{G}(\mathsf{input}@\mathsf{o}[j], \mathsf{k}')$, and Θ contains our restriction on the oracle queries and the induction hypothesis. We now use PRF on the last item of the frame and, to be able to conclude using the Fresh rule, show that the generated conditions are always true. We only show the condition for $\mathscr{P}$ as the one for $\mathscr{P}'$ is symmetrical and more easily proved. It intuitively expresses that the message hashed with G in $\mathsf{o}[j]$ cannot have been hashed with G anywhere else in $\mathsf{frame}@\mathsf{pred}(\mathsf{o}[j])$ and $\mathsf{input}@\mathsf{o}[j]$, i.e. at any step of the protocol execution before the output $\mathsf{o}[j]$:

$$j; \Theta \vdash \left[\left(\forall j'.\ \mathsf{o}[j'] < \mathsf{o}[j] \Rightarrow \mathsf{input}@\mathsf{o}[j'] \neq \mathsf{input}@\mathsf{o}[j]\right) \wedge \left(\forall i.\ \mathsf{a}[i] < \mathsf{o}[j] \Rightarrow s@\mathsf{a}[i] \neq \mathsf{input}@\mathsf{o}[j]\right)\right]_{\mathscr{P}}$$

The first conjunct is easily shown to be a consequence of $[\phi_{\text{rest}}]_{\mathscr{P}}$, and the second one is proved using the previous result, i.e. the strong secrecy of $s@\mathsf{a}[i]$ and using our new rule REACH-EQUIV.

A similar reasoning can be used to eliminate u_1 from our frames. The condition generated by PRF for $\mathscr{P}$ only differs slightly from what we had in the previous case (and the same goes for $\mathscr{P}'$):

$$j; \Theta \vdash \left[\left(\forall j'.\ \mathsf{o}[j'] < \mathsf{o}[j] \Rightarrow \mathsf{input}@\mathsf{o}[j'] \neq \mathsf{input}@\mathsf{o}[j]\right) \wedge \left(\forall i.\ \mathsf{a}[i] < \mathsf{o}[j] \Rightarrow s@\mathsf{pred}(\mathsf{a}[i]) \neq \mathsf{input}@\mathsf{o}[j]\right)\right]_{\mathscr{P}}$$

The two conjuncts are eliminated using the restriction on oracle queries, and the strong secrecy of $s@\mathsf{pred}(\mathsf{a}[i])$ which can be shown to be equal to $s@\mathsf{init}$ or some $s@\mathsf{a}[i']$.

Case where $\tau = \mathsf{a}[i]$. We also conclude using PRF. The generated condition is

$$i; \Theta \vdash \left[\left(\forall j.\ \mathsf{o}[j] < \mathsf{a}[i] \Rightarrow \mathsf{input}@\mathsf{o}[j] \neq s@\mathsf{a}[i]\right) \wedge \left(\forall i'.\ \mathsf{a}[i'] < \mathsf{a}[i] \Rightarrow s@\mathsf{a}[i'] \neq s@\mathsf{a}[i]\right)\right]_{\mathscr{P}}$$

and follows from the strong secrecy of $s@\mathsf{a}[i]$ for the first conjunct, and the non-repeating lemma for the second one.

6 Conclusion

We have presented an improvement of the theoretical foundations of the SQUIRREL prover, with a more natural model of protocol executions, memory cells, and a more expressive proof system. We have shown how these extensions enable security analyses that were previously out of scope.

As future work, we plan to finish implementing our generalized proof system, and the more precise PRF rule used in our case-study. A more open-ended direction of research is to study how the oracle restriction used in Section 5 could be proved in SQUIRREL. In order to show that for any execution violating the restriction, there exists another execution which satisfies it, we might turn to branching time semantics reasoning over more than one interleaving or at least more than one attacker computation. Finally, we could study the proof theoretical properties of our logic, e.g. the existence of normal forms.

References

[1] David Baelde, Stéphanie Delaune, Charlie Jacomme, Adrien Koutsos & Solène Moreau (2021): *An Interactive Prover for Protocol Verification in the Computational Model*. In: *SP 2021 - 42nd IEEE Symposium on Security and Privacy*, San Fransisco / Virtual, United States, p. t.b.d.

[2] Gergei Bana, Pedro Adão & Hideki Sakurada (2012): *Computationally Complete Symbolic Attacker in Action*. In: *FSTTCS*, *LIPIcs* 18, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 546–560, doi:10.4230/LIPIcs.FSTTCS.2012.546. Available at `https://doi.org/10.4230/LIPIcs.FSTTCS.2012.546`.

[3] Gergei Bana, Rohit Chadha & Ajay Kumar Eeralla (2018): *Formal Analysis of Vote Privacy Using Computationally Complete Symbolic Attacker*. In: *ESORICS (2)*, *LNCS* 11099, Springer, pp. 350–372.

[4] Gergei Bana & Hubert Comon-Lundh (2012): *Towards Unconditional Soundness: Computationally Complete Symbolic Attacker*. In: *POST*, *Lecture Notes in Computer Science* 7215, Springer, pp. 189–208.

[5] Gergei Bana & Hubert Comon-Lundh (2014): *A Computationally Complete Symbolic Attacker for Equivalence Properties*. In: *CCS*, ACM, pp. 609–620.

[6] Gilles Barthe, Benjamin Grégoire, Sylvain Heraud & Santiago Zanella Béguelin (2011): *Computer-Aided Security Proofs for the Working Cryptographer*. In: *CRYPTO*, *Lecture Notes in Computer Science* 6841, Springer, pp. 71–90.

[7] David A. Basin, Andreas Lochbihler & S. Reza Sefidgar (2020): *CryptHOL: Game-Based Proofs in Higher-Order Logic*. *J. Cryptology* 33(2), pp. 494–566.

[8] Bruno Blanchet (2006): *A Computationally Sound Mechanized Prover for Security Protocols*. In: *IEEE Symposium on Security and Privacy*, IEEE Computer Society, pp. 140–154.

[9] Mayla Brusò, Konstantinos Chatzikokolakis & Jerry den Hartog (2010): *Formal Verification of Privacy for RFID Systems*. In: *CSF*, IEEE Computer Society, pp. 75–88.

[10] Hubert Comon & Adrien Koutsos (2017): *Formal Computational Unlinkability Proofs of RFID Protocols*. In: *CSF*, IEEE Computer Society, pp. 100–114.

[11] Oded Goldreich, Shafi Goldwasser & Silvio Micali (1986): *How to construct random functions*. *J. ACM* 33(4), pp. 792–807.

[12] Adrien Koutsos (2019): *The 5G-AKA Authentication Protocol Privacy*. In: *EuroS&P*, IEEE, pp. 464–479.

[13] Guillaume Scerri & Ryan Stanley-Oakes (2016): *Analysis of Key Wrapping APIs: Generic Policies, Computational Security*. In: *CSF*, IEEE Computer Society, pp. 281–295.

[14] *The Squirrel Prover repository.* `https://github.com/squirrel-prover/squirrel-prover/`.

A Classical inference rules

We present some classical inference rules for reachability and general sequents in Figure 4.

Example inference rules for reachability sequents.

$$\frac{\Sigma, \alpha; \Theta : \Gamma \vdash_{\mathscr{P}} \phi}{\Sigma; \Theta : \Gamma \vdash_{\mathscr{P}} \forall \alpha.\phi} \qquad \frac{\Sigma; \Theta : \Gamma \vdash_{\mathscr{P}} \forall \tau.\phi}{\Sigma; \Theta : \Gamma \vdash_{\mathscr{P}} \phi\{\tau \to T\}} \qquad \frac{\Sigma; \Theta : \Gamma, \psi \vdash_{\mathscr{P}} \phi}{\Sigma; \Theta : \Gamma \vdash_{\mathscr{P}} \psi \Rightarrow \phi}$$

$$\frac{\Sigma; \Theta : \Gamma \vdash_{\mathscr{P}} \psi \qquad \Sigma; \Theta : \Gamma \vdash_{\mathscr{P}} \psi \Rightarrow \phi}{\Sigma; \Theta : \Gamma \vdash_{\mathscr{P}} \phi} \qquad \frac{\Sigma; \Theta : \Gamma, \neg\phi \vdash_{\mathscr{P}} \bot}{\Sigma; \Theta : \Gamma \vdash_{\mathscr{P}} \phi}$$

Inference rules for general sequents.

$$\frac{}{\Sigma; \Theta, F \vdash F} \qquad \frac{\Sigma; \Theta \vdash 0}{\Sigma; \Theta \vdash F} \qquad \frac{\Sigma; \Theta, F \to 0 \vdash 0}{\Sigma; \Theta \vdash F} \qquad \frac{\Sigma, \alpha; \Theta \vdash F}{\Sigma; \Theta \vdash \Pi\alpha.F} \qquad \frac{\Sigma; \Theta \vdash \Pi x.F}{\Sigma; \Theta \vdash F\{x \to t\}}$$

$$\frac{\Sigma; \Theta, F \vdash G}{\Sigma; \Theta \vdash F \to G} \qquad \frac{\Sigma; \Theta \vdash F \to G \qquad \Sigma; \Theta \vdash F}{\Sigma; \Theta \vdash G}$$

The introduction rules for $\forall$ and Π assume that $\alpha \notin \Sigma$. The elimination rule for Π is shown for sort message but has variants for sorts index and timestamp. The elimination rule for $\forall$ only has a variant for sort index — recall that quantification over messages is not allowed in local meta-formulas.

Figure 4: Classical reasoning for reachability and general sequents.