Cracking the Stateful Nut

Computational Proofs of Stateful Security Protocols using the SQUIRREL Proof Assistant

David Baelde*, Stéphanie Delaune*, Adrien Koutsos† and Solène Moreau*
* Univ Rennes, CNRS, IRISA, France
† Inria Paris, France

2022 IEEE 35th Computer Security Foundations Symposium (CSF) | 978-1-6654-8417-6/22/$31.00 ©2022 IEEE | DOI: 10.1109/CSF54842.2022.9919665

***Index Terms*—Security Protocols, Formal Methods, Logic, Computational Security, Interactive Theorem Proving.**

***Abstract*—Bana and Comon have proposed a logical approach to proving protocols in the computational model, which they call the Computationally Complete Symbolic Attacker (CCSA). The proof assistant SQUIRREL implements a verification technique that elaborates on this approach, building on a meta-logic over the CCSA base logic. In this paper, we show that this meta-logic can naturally be extended to handle protocols with mutable states (key updates, counters, *etc.*) and we extend SQUIRREL's proof system to be able to express the complex proof arguments that are sometimes required for these protocols. Our theoretical contributions have been implemented in SQUIRREL and validated on a number of case studies, including a proof of the YubiKey and YubiHSM protocols.**

I. INTRODUCTION

Formally specifying and verifying security protocols has become a major field of application for formal logic and automated reasoning. Decades of intensive research in that domain have lead to mature tools and industrial successes [1]–[3]. Some tools analyze protocols in so-called symbolic models [4]–[8] where attacker capabilities are represented by a fixed set of equations or inference rules over formal terms. Others deal with the cryptographer's computational model [9]–[11] where arbitrary probabilistic Turing machines compute over bitstrings. The symbolic models have enabled the earliest successes, generally allow for highly automated analyses, and can be used to discover actual attacks. However, it is very difficult to interpret a proof in a symbolic model in terms of computational attackers, as is witnessed by the limited practical impact of research on computational soundness [12]. Originally, many tools were designed to be fully automatic, e.g. [13], [14]. Newer tools, e.g. [4], [8], still have this objective but also incorporate interactivity, and even tools designed originally to be fully automated, e.g. PROVERIF, are moving towards it [15]. It is indeed an interesting compromise to tackle the analysis of complex security protocols. The computational model is more sought after than the symbolic one, as it provides stronger and more realistic security guarantees. But these come at a cost: it involves more complex notions and specific proof techniques, and mechanized proofs in that model often involve heavy user guidance. Yet there is no clear cut between the two approaches, and CRYPTOVERIF [9] in particular can sometimes derive proofs completely automatically.

In this paper, we consider a recent approach to proving protocols in the computational model, embodied in the proof assistant SQUIRREL [16]. It implements a meta-logic built on top of the logic introduced by Bana and Comon [17], which follows an approach called the computationally complete symbolic attacker (CCSA). The CCSA approach 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. To circumvent these problems, SQUIRREL builds on a meta-logic which 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 completely mechanized proofs of protocols using the CCSA approach [18].

The initial successes of the SQUIRREL system lie within a limited scope, in terms of the complexity of the protocols and of the proofs of their properties, and one naturally wonders how far the approach can be taken. In the present paper, we show how the SQUIRREL approach can be extended to support protocols with mutable states. Such situations are common: new keys may be derived regularly using cryptographic hash functions, counters or timestamps may be maintained to avoid replays, *etc.* At the semantic level, the meta-logic approach is easily adapted to incorporate states. At the level of proofs, however, significant improvements appear to be needed except for the simplest case studies.

- As we shall see, significant properties of stateful protocols often rely on a rich interplay between reachability and equivalence properties, which have to be proved simultaneously by mutual induction: *e.g.*, the strong secrecy (an equivalence property) of some state values at an instant T can be shown using the fact that these values never repeat (a reachability property), which itself relies on the strong secrecy of said state values at $T' < T$, *etc.*
- Moreover, because proofs of stateful protocols are more complex, they benefit more from automated deduction support. We found (bi)-deduction to be a useful notion in that regard. Roughly, the idea is that, if (part of) some messages of a protocol can be deduced by the adversary

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.

DOI 10.1109/CSF54842.2022.00018

from past messages, they can be safely ignored: indeed, they do not bring any new knowledge to the adversary.

Our main theoretical contribution is thus the introduction of a significant generalization of the sequent calculi of [18], allowing to capture security arguments mixing reachability and equivalence properties, and the design of a rich bi-deduction proof system, which can be used to automatically simplify some proof goals. On the practical side, we have implemented these improvements in an extension of SQUIRREL, and shown that they enable new case studies. In particular, we present proofs of the YubiKey and YubiHSM protocols, adapting in the computational model the TAMARIN development of [19]. Interestingly, the SQUIRREL development involves a similar amount of human effort than the TAMARIN analysis.

A full version of this paper including the omitted rules and proofs is available [20]. Our extension of SQUIRREL, as well as our case studies are available at:

https://squirrel-prover.github.io

Outline. We briefly recall the CCSA approach in Section II, before presenting in Section III the meta-logic given in [18] enriched to support mutable states, and sequences of terms. We show in Section IV some immediate applications of this extension, and also identify limitations of the proof system of [18]. This motivates the development of several improvements of this proof system, in Section V. We put everything together in Section VI where we present our case studies on the YubiKey and YubiHSM protocols. We conclude in Section VII.

II. BACKGROUND

The CCSA approach [17], [21] allows to reason about probabilistic polynomial-time computations using the simple symbolic setting of first-order logic. In a nutshell, terms are used to model computations, and a single predicate models indistinguishability of (sequences of) terms. Standard cryptographic assumptions can then be captured by axiom schemes, from which the indistinguishabilities of interest will have to be derived. In comparison with [17], we consider only the sort message, and a single attacker symbol att. For illustration purposes, we introduce below our running example.

Example 1. *We consider a variant of the OSK protocol [22], which is an RFID protocol.*

T *updates its state:* $\mathsf{s_T} := \mathsf{H}(\mathsf{s_T}, \mathsf{k})$
$T \to R : \mathsf{G}(\mathsf{s_T}, \mathsf{k}')$
$R \to T : \mathsf{ok}$ *if* $\mathsf{G}(\mathsf{s_T}, \mathsf{k}') = \mathsf{G}(\mathsf{H}(\mathsf{s_R}, \mathsf{k}), \mathsf{k}')$ *with* $\mathsf{s_R} \in DB$
R *updates the database* DB*:* $\mathsf{s_R} := \mathsf{H}(\mathsf{s_R}, \mathsf{k})$

We consider two keyed hash functions H *and* G *and assume that they both satisfy the* Pseudo Random Function *(PRF [23]) assumption. We will always use* H *with the key* k*, and* G *with* k′*. Each tag is associated to a mutable state* $\mathsf{s_T}$ *initialized with a secret value* s^0 *and can perform the following action any number of times: update* $\mathsf{s_T}$ *with* $\mathsf{H}(\mathsf{s_T}, \mathsf{k})$*, output* $\mathsf{G}(\mathsf{s_T}, \mathsf{k}')$*. We consider that readers have access to a database, containing an entry for each authorized tag. Each entry is initialized with the secret value* s^0 *of the corresponding tag. When receiving a message* x*, the reader looks up in the database for an entry* $\mathsf{s_R}$ *such that* $x = \mathsf{G}(\mathsf{H}(\mathsf{s_R}, \mathsf{k}), \mathsf{k}')$*. Upon success, the reader accepts the message and updates the entry* $\mathsf{s_R}$ *with* $\mathsf{H}(\mathsf{s_R}, \mathsf{k})$*. 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.*

A. 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 $\mathsf{if}\ \_\ \mathsf{then}\ \_\ \mathsf{else}\ \_$, and the constants empty, true and false.

Atomic formulas are of the form $u_1, \ldots, u_n \sim v_1, \ldots, v_n$ where $n \geq 1$ 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 protocol of Example 1, we use one name symbol* s_i^0 *for each tag* i*, two name symbols* k, k′ *for the keys, and two function symbols* H, G*. The term* $\mathsf{G}(\mathsf{H}(\mathsf{s}_i^0, \mathsf{k}), \mathsf{k}')$ *represents the message outputted at the first session of the tag* i*,* $\mathsf{G}(\mathsf{H}(\mathsf{H}(\mathsf{s}_i^0, \mathsf{k}), \mathsf{k}), \mathsf{k}')$ *the message outputted at the second session,* etc. *Assuming two different tags with respective initial secret values* s_1^0 *and* s_2^0*, the atomic formula*

$$\mathsf{G}(\mathsf{H}(\mathsf{s}_1^0, \mathsf{k}), \mathsf{k}'), \mathsf{G}(\mathsf{H}(\mathsf{s}_2^0, \mathsf{k}), \mathsf{k}') \\ \sim \mathsf{G}(\mathsf{H}(\mathsf{s}_1^0, \mathsf{k}), \mathsf{k}'), \mathsf{G}(\mathsf{H}(\mathsf{H}(\mathsf{s}_1^0, \mathsf{k}), \mathsf{k}), \mathsf{k}'),$$

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.

B. 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. Given a valuation σ mapping variables in t to Turing machines in $\mathbb{M}$'s domain, the interpretation $[\![t]\!]^\sigma_\mathbb{M}$ of a term as a Turing machine is defined as follows (we omit σ when t is ground, and write $[\![t]\!]$ when $\mathbb{M}$ is clear from the context).

Variables in $\mathcal{X}_B$ are interpreted using σ, and 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 ensures that syntactically distinct names are interpreted as *independent* random variables).

The symbols empty, true, false, EQ, $\langle \_, \_ \rangle$ and if _ then _ else _ are interpreted in the expected way. For instance, for any term t_1, t_2, $[\![\mathsf{EQ}(t_1, t_2)]\!]^\sigma_\mathbb{M}$ is the Turing machine that, on input $(1^\eta, \rho)$, returns 1 if $[\![t_1]\!]^\sigma_\mathbb{M}(1^\eta, \rho)$ and $[\![t_2]\!]^\sigma_\mathbb{M}(1^\eta, \rho)$ return the same result, and 0 otherwise. 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 the computational indistinguishability $\approx$, where two terms are computationally indistinguishable if no probabilistic PTIME machine can distinguish between them with non-negligible probability. More precisely, $d_1, \ldots, d_n \approx d'_1, \ldots, d'_n$ when for any PTIME Turing machine $\mathcal{A}$, the following quantity is negligible in η:

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

Here, $\mathbf{Pr}(\rho : \mathcal{M}(\rho) = 1)$ is the probability that the machine $\mathcal{M}$ accepts w.r.t. the random tapes in ρ. As it is standard with asymptotic security, a function is said to be negligible when it is asymptotically smaller than the inverse of any polynomial.

Given a computational model $\mathbb{M}$, a valuation σ and a formula ϕ, we write $\mathbb{M}, \sigma \models \phi$ when ϕ, with its variables interpreted by σ, is satisfied in the computational model $\mathbb{M}$. We say that ϕ is valid when $\mathbb{M}, \sigma \models \phi$ for all $\mathbb{M}$ and σ.

Example 3. *Let n and m be two distinct names. The formulas $\mathsf{n} \sim \mathsf{m}$ and $\mathsf{EQ}(\mathsf{n}, \mathsf{m}) \sim \mathsf{false}$ are valid, as no attacker can distinguish between two uniform random samplings of the same length, and two such samplings that are independent have a negligible probability of being equal.*

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. Some axioms are sound in all computational models, *e.g.* the symmetry of $\sim$, or properties of if _ then _ else _. Other axioms reflect cryptographic assumptions on the security primitives, such as the PRF assumption for hash functions. Such axioms exclude any computational model that does not satisfy the assumptions under which a security protocol has been designed.

C. Limitations of the base logic approach

As mentioned in the introduction, this approach has some limitations. First, it can only deal with bounded executions, *e.g.* the equivalence formula in Example 2 expresses the unlinkability of only two sessions of the protocol. Second, there is no support for proof mechanization. The work of [18] elaborates on the CCSA approach to solve these limitations, by designing a meta-logic over the base logic, a proof system for this meta-logic and by implementing them in a new interactive prover SQUIRREL. Nevertheless, the framework of [18] does not support stateful protocols. In the next section, we extend the meta-logic syntax and semantics with mutable states, and show in Section IV some direct applications of our extension.

III. META-LOGIC

We now introduce the syntax and the semantics of our meta-logic, which extends the meta-logic of [18] with memory cells, to model stateful protocols; and sequences, which are useful to reason about sets of terms. In order to match the intuitive notion of a trace, we also add a new atomic formula $\mathsf{happens}(T)$, expressing that T occurs in the execution.

A. Meta-logic 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* which are used to identify an element, *e.g.* a session or an item in a database. We consider infinite sets $\mathcal{T}$, $\mathcal{X}$, and $\mathcal{I}$ of variables of each sort.

We assume a set $\mathcal{F}$ of indexed function symbols to represent protocol functions and cryptographic primitives (*e.g.* encryptions or keyed hash functions). Function symbols representing cryptographic primitives have index arity 0, and a message arity depending on the primitive. In contrast, function symbols representing identities have 0 as message arity and an arbitrary index arity. We also assume a set $\mathcal{N}$ of indexed name symbols to model random samplings, and a set of indexed action symbols $\mathcal{A}$ to model particular time points.

Given a meta-logic signature $\Sigma = (\mathcal{F}, \mathcal{N}, \mathcal{A}, \mathcal{S})$ and sets of variables $\mathcal{T}$, $\mathcal{X}$, and $\mathcal{I}$, we give in Figure 1 the syntax of meta-terms of sorts timestamp (noted T) and message (noted t), and of local meta-formulas (noted ϕ). Note that the notion of *local* meta-formulas will take all its meaning in Section V where we will introduce *global* meta-formulas. Meta-terms of sort index are restricted to variables $i \in \mathcal{I}$. We denote by $\mathsf{fv}(\_)$ the free variables (of any sort) of a meta-term or meta-formula.

Messages are modeled using terms of sort message, relying on *macros* to represent specific terms involved in the protocol. Since these values change throughout the protocol execution,

$$
\begin{array}{rcll}
T & := & \tau & \textit{(variable, } \tau \in \mathcal{T}\textit{)} \\
& \mid & \mathsf{a}[i_1, \ldots, i_k] & \textit{(action, } \mathsf{a} \in \mathcal{A}\textit{)} \\
& \mid & \mathsf{init} \mid \mathsf{pred}(T) & \\
t & := & x & \textit{(variable, } x \in \mathcal{X}\textit{)} \\
& \mid & \mathsf{s}[i_1, \ldots, i_k]@T & \textit{(memory cell, } \mathsf{s} \in \mathcal{S}\textit{)} \\
& \mid & \mathsf{n}[i_1, \ldots, i_k] & \textit{(name, } \mathsf{n} \in \mathcal{N}\textit{)} \\
& \mid & \mathsf{f}[i_1, \ldots, i_k](t_1, \ldots, t_n) & \textit{(function, } \mathsf{f} \in \mathcal{F}\textit{)} \\
& \mid & \mathsf{input}@T \mid \mathsf{output}@T \mid \mathsf{frame}@T & \textit{(builtin macros)} \\
& \mid & \mathsf{if}\ \phi\ \mathsf{then}\ t\ \mathsf{else}\ t' & \textit{(conditional)} \\
& \mid & \mathsf{find}\ \vec{i}\ \mathsf{suchthat}\ \phi\ \mathsf{in}\ t\ \mathsf{else}\ t' & \textit{(index lookup)} \\
& \mid & \mathsf{seq}(\vec{\alpha} : t) & \textit{(sequence of terms)} \\
\alpha & := & i \mid \tau & \textit{(variable, } i \in \mathcal{I},\ \tau \in \mathcal{T}\textit{)} \\
A & := & t = t' \mid i = i' \mid T = T' \mid T < T' \mid T \leq T' & \\
& \mid & \mathsf{happens}(T) \mid \mathsf{cond}@T \mid \mathsf{exec}@T & \\
\phi & := & A \mid \top \mid \bot \mid \phi \wedge \phi' \mid \phi \vee \phi' \mid \phi \Rightarrow \phi' \mid \neg\phi & \\
& \mid & \forall i.\phi \mid \exists i.\phi \mid \forall \tau.\phi \mid \exists \tau.\phi &
\end{array}
$$

Syntactic sugar: $(\mathsf{if}\ \phi\ \mathsf{then}\ t) \overset{\text{def}}{=} \mathsf{if}\ (\phi\ \mathsf{then}\ t\ \mathsf{else}\ \mathsf{empty})$ and $\mathsf{seq}(\vec{\alpha} : t \mid \phi) \overset{\text{def}}{=} \mathsf{seq}(\vec{\alpha} : \mathsf{if}\ \phi\ \mathsf{then}\ t)$.

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

macros are applied to a timestamp term that indicates when the macro is evaluated. Our logic features several predefined macros: $\mathsf{input}@T$ and $\mathsf{output}@T$ refer to the input and output messages of the action executed at time point T; $\mathsf{cond}@T$ and $\mathsf{exec}@T$ encode, resp., the condition of the action T, and the conjunction of all the conditions to reach T; and $\mathsf{frame}@T$ represents all the messages known by the attacker at time point T. The special timestamp constant init stands for the initial time point, and we rely on pred to denote the predecessor of a given time point T. Finally, pred^n is used as a shortcut for n successive applications of pred. We assume a set $\mathcal{S}$ of indexed state macro symbols to represent memory cells. For instance, if s is a state macro symbol of arity 1, the message $\mathsf{s}[i]@T$ refers to the contents of $\mathsf{s}[i]$ at time T. 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.

Lastly, we often use sequences when reasoning on protocols, *e.g.* to enrich equivalences with collections of messages that are already known to the attacker or can safely be disclosed as far as the property being proved is concerned. Once meta-interpreted in a given trace model, a sequence $\mathsf{seq}(\vec{\alpha} : t)$ is simply a list built as nested pairs. However, we need to reason on these objects in the meta-logic, uniformly for all trace models, which requires the use of a specific construct.

Example 4. *In Example 2, we explained how to model the messages outputted by the tag in the base logic. In particular, we had to consider as many names s^0_i as tag's identities. In the meta-logic, we can use indexed names to model unbounded collections of objects. We thus assume a name $\mathsf{s}^0 \in \mathcal{N}$ of index arity 1, such that $\mathsf{s}^0[i]$ models the initial secret value associated to the tag i. We also take two state macro symbols, $\mathsf{s_T}$ and $\mathsf{s_R}$ of index arity 1, modeling the current value of the memory cell and the database entry associated to a tag. The message outputted by a tag i at time T is then modeled by the meta-term $\mathsf{G}(\mathsf{H}(\mathsf{s_T}[i]@\mathsf{pred}(T), \mathsf{k}), \mathsf{k'})$. In this message, $\mathsf{s_T}[i]@\mathsf{pred}(T)$ is the value of the memory cell just before time point T,* i.e. *a (possibly empty) stack of hashes $\mathsf{H}(\ldots(\mathsf{H}(\mathsf{s}^0[i]), \mathsf{k}), \ldots), \mathsf{k})$. We finally model as a (local) meta-formula the condition under which a reader executing at time T' will accept the input as coming from tag i:*

$$\phi^{T'}_{\mathsf{reader}} \overset{def}{=} \mathsf{input}@T' = \mathsf{G}(\mathsf{H}(\mathsf{s_R}[i]@\mathsf{pred}(T'), \mathsf{k}), \mathsf{k'}).$$

The example below illustrates the use of the $\mathsf{seq}(\vec{\alpha} : t)$ construct.

Example 5. *Continuing our running example, we may want to consider the sequence of terms corresponding to the successive values stored in the memory cell $\mathsf{s_T}[i]$ of each tag i during a valid execution. Such a sequence can be represented by the meta-term:*

$$\mathsf{seq}(\tau, i : \mathsf{s_T}[i]@\tau \mid \mathsf{happens}(\tau) \wedge \mathsf{exec}@\tau)$$

If we only want to consider the successive values stored in the memory cell of the tag i_0, we will consider the meta-term

- $\mathsf{seq}(\tau : \mathsf{s_T}[i_0]@\tau \mid \mathsf{happens}(\tau) \wedge \mathsf{exec}@\tau)$*; or*
- $\mathsf{seq}(\tau, i : \mathsf{s_T}[i]@\tau \mid i = i_0 \wedge \mathsf{happens}(\tau) \wedge \mathsf{exec}@\tau)$

B. Protocols

Our local meta-logic formulas are interpreted as boolean terms of the base logic, depending on an execution of 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

$$\mathsf{a}[\vec{i}].(\phi_{\mathsf{a}[\vec{i}]}, o_{\mathsf{a}[\vec{i}]}, \{\mathsf{s}[\vec{j}] \leftarrow u_{\mathsf{a}[\vec{i}],\mathsf{s}[\vec{j}]} \mid \mathsf{s} \in \mathcal{S}\})$$

is formed from an action symbol a, distinct index variables $\vec{i}$, a local meta-logic formula $\phi_{\mathsf{a}[\vec{i}]}$, a meta-logic term $o_{\mathsf{a}[\vec{i}]}$ of sort message, and for each state macro symbol $\mathsf{s} \in \mathcal{S}$ and distinct index variables $\vec{j}$ (disjoint from $\vec{i}$), a meta-logic term $u_{\mathsf{a}[\vec{i}],\mathsf{s}[\vec{j}]}$ of sort message. We require that

$$\mathsf{fv}(\phi_{\mathsf{a}[\vec{i}]}, o_{\mathsf{a}[\vec{i}]}) \subseteq \{\vec{i}\} \quad \textit{and} \quad \mathsf{fv}(u_{\mathsf{a}[\vec{i}],\mathsf{s}[\vec{j}]}) \subseteq \{\vec{i}, \vec{j}\}.$$

The formula $\phi_{\mathsf{a}[\vec{i}]}$ is called the condition *of the action, $o_{\mathsf{a}[\vec{i}]}$ its* output *and $\{u_{\mathsf{a}[\vec{i}],\mathsf{s}[\vec{j}]} \mid s \in \mathcal{S}\}$ its state update terms.*

An action $\mathsf{a}[\vec{i}].(\phi_{\mathsf{a}[\vec{i}]}, o_{\mathsf{a}[\vec{i}]}, \{\mathsf{s}[\vec{j}] \leftarrow u_{\mathsf{a}[\vec{i}],\mathsf{s}[\vec{j}]} \mid \mathsf{s} \in \mathcal{S}\})$ models that, if $\phi_{\mathsf{a}[\vec{i}]}$ holds, $o_{\mathsf{a}[\vec{i}]}$ may be emitted, after having updated each memory cell $\mathsf{s}[\vec{j}]$ with the value $u_{\mathsf{a}[\vec{i}],\mathsf{s}[\vec{j}]}$. Conditional branching may be modeled using two actions, the condition of the second action being the negation of the first action's condition.

A protocol is a set of actions equipped with a dependency relation, which constrains the order of execution of actions. We require that an action only refers to timestamps of actions occurring strictly before it, except when referring to its own input or to the value of a state macro at the current time if it is in the output term (*i.e.* after all updates have been performed).

Definition 2. *Given a finite set $\mathcal{A}$ of action symbols and a finite set $\mathcal{S}$ of state macro symbols, a protocol $\mathcal{P} = (\mathcal{P}_{\mathsf{act}}, \mathcal{U}_0, <)$ over $(\mathcal{A}, \mathcal{S})$ is a finite set $\mathcal{U}_0 \overset{def}{=} \{\mathsf{s}[\vec{j}] \leftarrow u_{0,\mathsf{s}[\vec{j}]} \mid \mathsf{s} \in \mathcal{S}\}$ of initial state values, one for each state macro symbol, and a finite set $\mathcal{P}_{\mathsf{act}}$ 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 the following conditions.*

- *For each memory cell $\mathsf{s}[\vec{j}]$, $u_{0,\mathsf{s}[\vec{j}]}$ is a macro-free meta-logic term of sort message with free variables among $\vec{j}$.*
- *The partial order $<$ is insensitive to the choice of specific index variables,* i.e. *we have $\mathsf{a}_1[\vec{i_1}] < \mathsf{a}_2[\vec{i_2}]$ if, and only if, $\mathsf{a}_1[\sigma(\vec{i_1})] < \mathsf{a}_2[\sigma(\vec{i_2})]$ for any $\mathsf{a}_1, \mathsf{a}_2, \vec{i_1}$ and $\vec{i_2}$ and for any bijective variable renaming $\sigma : \mathcal{I} \rightarrow \mathcal{I}$.*
- *Actions only refer to information about previously executed actions. For every action $\mathsf{a}[\vec{i}].(\_) \in \mathcal{P}_{\mathsf{act}}$:*
  - *each subterm T of sort timestamp occurring in the condition $\phi_{\mathsf{a}[\vec{i}]}$ or in an update $u_{\mathsf{a}[\vec{i}],\mathsf{s}[\vec{j}]}$ is either (i) of the form $\mathsf{a}_0[\vec{i_0}]$ with $\mathsf{a}_0[\vec{i_0}] < \mathsf{a}[\vec{i}]$, or (ii) of the form* $\mathsf{pred}^n(\mathsf{a}[\vec{i}])$ *with $n \geq 1$, or (iii) of the form $\mathsf{a}[\vec{i}]$ and is applied to an input macro,*
  - *each subterm T of sort timestamp occurring in $o_{\mathsf{a}[\vec{i}]}$ either verifies (i), (ii) or (iii); or (iv) is of the form $\mathsf{a}[\vec{i}]$ and is applied to a state macro.*

Example 6. *Coming back to our running example, let $\mathsf{T} \in \mathcal{A}$ be an action symbol of index arity 2. The action $\mathsf{T}[i,j].(\_)$ defined below models the output performed by the tag, as well as the way the memory cells are updated. The memory cell $\mathsf{s_T}[i]$ is updated with $\mathsf{H}(\mathsf{s_T}[i]@\mathsf{pred}(\mathsf{T}[i,j]), \mathsf{k})$, whereas $\mathsf{s_T}[i']$ with $i' \neq i$, and $\mathsf{s_R}[i']$, are left unchanged.*

$$\begin{aligned}
\mathsf{T}[i,j].(&\mathsf{true}, \mathsf{G}(\mathsf{H}(\mathsf{s_T}[i]@\mathsf{pred}(\mathsf{T}[i,j])), \mathsf{k}), \mathsf{k}'),\\
&\{\mathsf{s_T}[i'] \leftarrow \mathsf{if}\ i' = i\ \mathsf{then}\ \mathsf{H}(\mathsf{s_T}[i]@\mathsf{pred}(\mathsf{T}[i,j]), \mathsf{k})\\
&\qquad\qquad\qquad\qquad \mathsf{else}\ \mathsf{s_T}[i']@\mathsf{pred}(\mathsf{T}[i,j]),\\
&\quad \mathsf{s_R}[i'] \leftarrow \mathsf{s_R}[i']@\mathsf{pred}(\mathsf{T}[i,j])\})
\end{aligned}$$

We similarly define the action $\mathsf{R}[j,i].(\_)$ of a reader session j accepting tag i, and $\mathsf{R}_1[j].(\_)$ of a reader session j rejecting its input. We take $\phi_{\mathsf{R}[j,i]} \overset{def}{=} \phi^{\mathsf{R}[j,i]}_{\mathsf{reader}}$ from Example 4, and $o_{\mathsf{R}[j]} \overset{def}{=}$ ok. *It remains to model the updates:*

$$\begin{aligned}
\mathsf{s_R}[i'] &\leftarrow \mathsf{if}\ i' = i\ \mathsf{then}\ \mathsf{H}(\mathsf{s_R}[i]@\mathsf{pred}(\mathsf{R}[j,i]), \mathsf{k})\\
&\qquad\quad \mathsf{else}\ \mathsf{s_R}[i']@\mathsf{pred}(\mathsf{R}[j,i])\\
\mathsf{s_T}[i'] &\leftarrow \mathsf{s_T}[i']@\mathsf{pred}(\mathsf{R}[j,i])
\end{aligned}$$

For the failure action $\mathsf{R}_1[j]$ we take $\phi_{\mathsf{R}_1[j]} \overset{def}{=} \forall i. \neg\phi^{\mathsf{R}_1[j]}_{\mathsf{reader}}$, $o_{\mathsf{R}_1[j]} \overset{def}{=}$ empty, *and updates that leave cells unchanged. To complete the protocol modeling our running example we initialize cells with s^0 and specify an empty partial order:*

$$\begin{aligned}
\mathcal{P}_{\mathsf{ex}} \overset{def}{=} (\{&\mathsf{T}[i,j].(\_), \mathsf{R}[j,i].(\_), \mathsf{R}_1[j].(\_)\},\\
&\{\mathsf{s_T}[i] \leftarrow \mathsf{s}^0[i], \mathsf{s_R}[i] \leftarrow \mathsf{s}^0[i]\}, \emptyset).
\end{aligned}$$

C. 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 in the notion of *trace model*, which we are now going to define.

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

Definition 3. *Given a set $\mathcal{A}$ of action symbols, a* concrete action *is the application of an action symbol $\mathsf{a} \in \mathcal{A}$ to k* integers *(k is the index arity of a). We lift the partial order of a protocol $\mathcal{P}$ to concrete actions. For any $\sigma : \mathcal{I} \rightarrow \mathbb{N}$, we have that $\mathsf{a}[\sigma(i_1), \ldots, \sigma(i_k)] < \mathsf{b}[\sigma(j_1), \ldots, \sigma(j_l)]$ when $\mathsf{a}[i_1, \ldots, i_k] < \mathsf{b}[j_1, \ldots, j_l]$.*

We define next the possible *interleavings* of actions for a given protocol, which over-approximates the actual possible executions by taking only dependency constraints into account.

Definition 4. *Given a protocol $\mathcal{P}$, an* interleaving *is a sequence of concrete actions $\alpha_1 \ldots \alpha_n$ 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 $\beta = \alpha_j$.*

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

In order to interpret meta-terms and local 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 $\mathcal{P}$) 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;*
- *$\mathcal{D}_\mathcal{T}$ contains two special symbols* init, undef, *and a subset of $\mathcal{D}_a \overset{def}{=} \{\mathsf{a}[k_1, \ldots, k_n] \mid \mathsf{a} \in \mathcal{A}, k_1, \ldots, k_n \in \mathcal{D}_\mathcal{I}\}$;*
- *$<_\mathcal{T}$ is a total ordering on $\mathcal{D}_\mathcal{T} \smallsetminus \{\mathsf{undef}\}$ such that* init *is minimal, and such that the sequence of elements of $\mathcal{D}_\mathcal{T} \smallsetminus \{\mathsf{undef}\}$ ordered by $<_\mathcal{T}$ is an interleaving of $\mathcal{P}$;*
- *$\sigma_\mathcal{I} : \mathcal{I} \rightarrow D_\mathcal{I}$ and $\sigma_\mathcal{T} : \mathcal{T} \rightarrow D_\mathcal{T}$ are mappings that interpret index and timestamp variables as elements of their respective domains.*

Given a trace model, we define a predecessor function $\mathsf{pred}_\mathcal{T} : \mathcal{D}_\mathcal{T} \rightarrow \mathcal{D}_\mathcal{T}$ which maps undef to itself, init to undef, and all other elements v to the largest element v' such that $v' <_\mathcal{T} v$. Moreover, we denote by $\leq_\mathcal{T}$ the reflexive closure of $<_\mathcal{T}$ on $\mathcal{D}_\mathcal{T} \smallsetminus \{\mathsf{undef}\}$. When $\mathbb{T} = (\mathcal{D}_\mathcal{I}, \mathcal{D}_\mathcal{T}, <_\mathcal{T}, \sigma_\mathcal{I}, \sigma_\mathcal{T})$

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

Example 7. *Consider the protocol $\mathcal{P}_{\mathsf{ex}}$ introduced in Example 6. A possible trace model $\mathbb{T}$ associated to this protocol is the tuple $(\mathcal{D}_\mathcal{I}, \mathcal{D}_\mathcal{T}, <_\mathcal{T}, \sigma_\mathcal{I}, \sigma_\mathcal{T})$ where:*

- $\mathcal{D}_\mathcal{I} \overset{def}{=} \{1, 2, 3\}$;
- $\mathcal{D}_\mathcal{T} \overset{def}{=} \{\mathsf{undef}, \mathsf{init}, \mathsf{T}[1,3], \mathsf{T}[2,1], \mathsf{T}[2,2], \mathsf{R}[1,2], \mathsf{R}_1[3]\}$;
- *$<_\mathcal{T}$ is a total ordering such that:* $\mathsf{init} < \mathsf{T}[2,1] < \mathsf{T}[1,3] < \mathsf{R}[1,2] < \mathsf{T}[2,2] < \mathsf{R}_1[3]$;
- *$\sigma_\mathcal{I}$ and $\sigma_\mathcal{T}$ are mappings that interpret index and timestamp variables to $\mathcal{D}_\mathcal{I}$ and $\mathcal{D}_\mathcal{T}$ respectively. For illustration purposes, we consider that $\sigma_\mathcal{I}(i) = \sigma_\mathcal{I}(j) = 2$.*

We have $\mathsf{pred}_\mathcal{T}(\mathsf{R}_1[3]) = \mathsf{T}[2,2]$, $\mathsf{pred}_\mathcal{T}(\mathsf{T}[2,2]) = \mathsf{R}[1,2]$, *and so on.*

2) Translation: We can now give our translation for meta-terms and local meta-formulas. This translation is similar to the one introduced in [18], but adapted to our notion of trace model in which some actions are allowed to not happen. We recall its general principle while highlighting the main differences with the translation given in [18].

The translation $(\_)^\mathbb{T}_\mathcal{P}$ is parameterized by a protocol $\mathcal{P}$ and a trace model $\mathbb{T}$ of $\mathcal{P}$. First, as expected, we have that $(\tau)^\mathbb{T}_\mathcal{P} = \sigma_\mathcal{T}(\tau)$, $(\mathsf{init})^\mathbb{T}_\mathcal{P} = \mathsf{init}$, and $(\mathsf{pred}(T))^\mathbb{T}_\mathcal{P} = \mathsf{pred}_\mathcal{T}((T)^\mathbb{T}_\mathcal{P})$. 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 $\mathsf{n}_{\sigma_\mathcal{I}(i_1), \ldots, \sigma_\mathcal{I}(i_p)}$. For some constructions, it is a bit more complicated. For instance, the sequence construct is not available in the base logic, but is translated using nested pairs. As an example, we show the interpretation of a sequence over a single index variable i.

$$(\mathsf{seq}(i \colon t))^\mathbb{T}_\mathcal{P} \overset{\text{def}}{=} \langle (t)^{\mathbb{T}[i \mapsto k_1]}_\mathcal{P}, \langle \ldots,\ (t)^{\mathbb{T}[i \mapsto k_n]}_\mathcal{P} \rangle \ldots \rangle$$

where $k_1, \ldots, k_n$ is an enumeration of $D_\mathcal{I}$. Similarly, the interpretation of (local) meta-formulas is quite straightforward using finite boolean expressions to translate quantifications over index and timestamp variables. For instance, we have that:

$$(\forall i.\phi)^\mathbb{T}_\mathcal{P} \quad \overset{\text{def}}{=} \quad \dot{\wedge}_{k \in D_\mathcal{I}}\ (\phi)^{\mathbb{T}\{i \mapsto k\}}_\mathcal{P}$$

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

- $(T = T')^\mathbb{T}_\mathcal{P} = \mathsf{true}$ iff $(T)^\mathbb{T}_\mathcal{P} = (T')^\mathbb{T}_\mathcal{P}$;
- $(T < T')^\mathbb{T}_\mathcal{P} = \mathsf{true}$ iff $(T)^\mathbb{T}_\mathcal{P} <_\mathcal{T} (T')^\mathbb{T}_\mathcal{P}$;
- $(T \leq T')^\mathbb{T}_\mathcal{P} = \mathsf{true}$ iff $(T)^\mathbb{T}_\mathcal{P} \leq_\mathcal{T} (T')^\mathbb{T}_\mathcal{P}$;
- $(\mathsf{happens}(T))^\mathbb{T}_\mathcal{P} = \mathsf{true}$ iff $(T)^\mathbb{T}_\mathcal{P} \in \mathcal{D}_\mathcal{T} \smallsetminus \{\mathsf{undef}\}$.

Lastly, we give in Figure 2 the interpretation of macros. Roughly, an output macro is replaced by the meta-term as specified by the protocol, and is then interpreted in the trace model $\mathbb{T}$. The cond macro has a similar treatment and produces a base logic formula corresponding to the condition 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, 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.

$$(\mathsf{s}[\vec{l}]@T)^\mathbb{T}_\mathcal{P} \overset{\text{def}}{=}$$

$$\begin{cases} \mathsf{empty} & \text{if } (T)^\mathbb{T}_\mathcal{P} = \mathsf{undef} \\ (u_{0,\mathsf{s}[\vec{j}]})^{\mathbb{T}\{\vec{j} \mapsto \vec{k}_0\}}_\mathcal{P} & \text{if } (T)^\mathbb{T}_\mathcal{P} = \mathsf{init}, (\vec{l})^\mathbb{T}_\mathcal{P} = \vec{k}_0 \\ & \text{and } (\mathsf{s}[\vec{j}] \leftarrow u_{0,\mathsf{s}[\vec{j}]}) \in \mathcal{U}_0 \\ (u_{\mathsf{a}[\vec{i}],\mathsf{s}[\vec{j}]})^{\mathbb{T}\{\vec{i} \mapsto \vec{k}, \vec{j} \mapsto \vec{k}_0\}}_\mathcal{P} & \text{if } (T)^\mathbb{T}_\mathcal{P} = \mathsf{a}[\vec{k}] \in \mathcal{D}_\mathcal{T}, (\vec{l})^\mathbb{T}_\mathcal{P} = \vec{k}_0 \end{cases}$$

$$(m@T)^\mathbb{T}_\mathcal{P} \overset{\text{def}}{=}$$

$$\begin{cases} \mathsf{empty} & \text{if } (T)^\mathbb{T}_\mathcal{P} = \mathsf{undef} \\ & \text{and } m \in \{\mathsf{input}, \mathsf{output}, \mathsf{frame}\} \\ \mathsf{false} & \text{if } (T)^\mathbb{T}_\mathcal{P} = \mathsf{undef} \text{ and } m \in \{\mathsf{cond}, \mathsf{exec}\} \\ (m_{\mathsf{init}})^\mathbb{T}_\mathcal{P} & \text{if } (T)^\mathbb{T}_\mathcal{P} = \mathsf{init} \\ (m_{\mathsf{a}[\vec{i}]})^{\mathbb{T}\{\vec{i} \mapsto \vec{k}\}}_\mathcal{P} & \text{if } (T)^\mathbb{T}_\mathcal{P} = \mathsf{a}[\vec{k}] \in \mathcal{D}_\mathcal{T} \end{cases}$$

where m_{init} and $m_{\mathsf{a}[\vec{i}]}$ are defined as follows:

$$\mathsf{cond}_{\mathsf{init}} = \mathsf{exec}_{\mathsf{init}} = \mathsf{true}$$

$$\mathsf{input}_{\mathsf{init}} = \mathsf{frame}_{\mathsf{init}} = \mathsf{output}_{\mathsf{init}} = \mathsf{empty}$$

$$\mathsf{output}_{\mathsf{a}[\vec{i}]} = o_{\mathsf{a}[\vec{i}]} \quad \mathsf{exec}_{\mathsf{a}[\vec{i}]} = \mathsf{cond}@\mathsf{a}[\vec{i}] \wedge \mathsf{exec}@\mathsf{pred}(\mathsf{a}[\vec{i}])$$

$$\mathsf{cond}_{\mathsf{a}[\vec{i}]} = \phi_{\mathsf{a}[\vec{i}]} \quad \mathsf{input}_{\mathsf{a}[\vec{i}]} = \mathsf{att}\big(\mathsf{frame}@\mathsf{pred}(\mathsf{a}[\vec{i}])\big)$$

$$\begin{aligned} \mathsf{frame}_{\mathsf{a}[\vec{i}]} = \langle &\mathsf{exec}@\mathsf{a}[\vec{i}], \langle \mathsf{if}\ \mathsf{exec}@\mathsf{a}[\vec{i}]\ \mathsf{then}\ \mathsf{output}@\mathsf{a}[\vec{i}] \\ &\mathsf{else}\ \mathsf{empty}, \mathsf{frame}@\mathsf{pred}(\mathsf{a}[\vec{i}])\rangle\rangle \end{aligned}$$

Fig. 2. Interpretation of macros.

Example 8. *Considering the trace model of Example 7 on our running example, we have the following translation:*

$$\begin{aligned} (\mathsf{output}@\mathsf{T}[i,j])^\mathbb{T}_{\mathcal{P}_{\mathsf{ex}}} &= (o_{\mathsf{T}[i,j]})^\mathbb{T}_{\mathcal{P}_{\mathsf{ex}}} \\ &= (\mathsf{G}(\mathsf{H}(\mathsf{s}_\mathsf{T}[i]@\mathsf{pred}(\mathsf{T}[i,j]), \mathsf{k}), \mathsf{k}'))^\mathbb{T}_{\mathcal{P}_{\mathsf{ex}}} \\ &= \mathsf{G}(\mathsf{H}((\mathsf{s}_\mathsf{T}[i]@\mathsf{pred}(\mathsf{T}[i,j]))^\mathbb{T}_{\mathcal{P}_{\mathsf{ex}}}, \mathsf{k}), \mathsf{k}'). \end{aligned}$$

We slightly abuse notation by using the same function symbols G, H, k *and* k' *in the base logic and the meta-logic.*

Then, the translation $(\mathsf{s}_\mathsf{T}[i]@\mathsf{pred}(\mathsf{T}[i,j]))^\mathbb{T}_{\mathcal{P}_{\mathsf{ex}}}$ is unfolded to a stack of hashes until reaching the initial time point init. *Here, after some basic reasoning in the base logic regarding the* if _ then _ else *construct, we obtain:*

$$(\mathsf{s}_\mathsf{T}[i]@\mathsf{pred}(\mathsf{T}[i,j]))^\mathbb{T}_{\mathcal{P}_{\mathsf{ex}}} = \mathsf{H}(\mathsf{s}^0_2, \mathsf{k}).$$

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

Example 9. *The following meta-formula models an authentication property which is valid in any computational model interpreting* G *and* H *as* EUF-CMA *secure hash functions.*

$$\forall j, i.\ \mathsf{happens}(R[j,i]) \Rightarrow \mathsf{cond}@R[j,i] \Rightarrow$$
$$(\exists j'. T[i,j'] < R[j,i] \wedge \mathsf{output}@T[i,j'] = \mathsf{input}@R[j,i])$$

This formula states that, whenever the condition of the reader's action $R[j,i]$ *holds, there must be some session* j' *of the tag* i *(which uses the seed* $\mathsf{s}_0[i]$ *to initialize its memory cell) that has been executed before* $R[j,i]$*. Moreover, the output of the tag's action coincides with the input of the reader's action. Here, we use quantifications to express our security goal. In the base logic, we would have to first fix a trace model, and then to explicitly enumerate all possible values for* j, i*, and* j'*.*

Comparison with [18]: In the notion of trace model of [18], the set $\mathcal{D}_\mathcal{T}$ contains init and *all* concrete actions of the set $\mathcal{D}_a$ (see Definition 5). In our new definition, $\mathcal{D}_\mathcal{T}$ contains only a subset of all possible concrete actions, which correspond to the actions that happen. For example, this change allows to model conflicts between actions: for instance, the axiom $\forall \vec{i}, \vec{j}.\ \neg(\mathsf{happens}(\mathsf{a}_1[\vec{i}]) \wedge \mathsf{happens}(\mathsf{a}_2[\vec{j}]))$ rules out traces where the two actions are scheduled.

IV. Applications and limitations

We present in Table I some case studies corresponding to protocols with global mutable states. These case studies use the extension of the meta-logic presented above to support mutable states, but rely on the same proof system as [18]. For each protocol, we give the security properties that we proved and the intermediate lemmas (also proved with SQUIRREL) used to carry out the proofs. We will comment below on these lemmas. All files are between 100 and 500 lines long (for both modeling and proof script). The length of each development mostly depends on the number of intermediate lemmas used.

Protocol	**Intermediate Lemmas**	**Security Properties**
Toy Counter [15]	Counter increase	Secrecy
CANAuth [15]	Counter increase	Authentication, Injectivity
Running example	Last update, Disjoint chains	Authentication
SLK06 [24]	-	Authentication
YPLRK05 [24]	No update	Authentication

TABLE I
CASE STUDIES OF SOME STATEFUL PROTOCOLS

1) Protocols with counters: We first consider two stateful protocols from [15], which rely on counter values that are incremented throughout the protocols execution. Toy Counter is a toy protocol where two agents A and B access a shared counter. We prove that this protocol provides some form of secrecy (expressed as a reachability property). CANAuth is a message authentication protocol where every agent, who can be both initiator and responder, stores a counter in memory which is incremented at each action of the agent. Proving the security of these protocols required reasoning on counter values. To that end, we axiomatized the counters ordering relation in SQUIRREL. This can be done using, *e.g.*, the axioms (i.e. global meta-formulas) in Listing 1.

```
abstract Succ : message → message
abstract (~<) : message→ message→ boolean

axiom orderTrans (n1,n2,n3:message):
   n1 ~< n2 ⇒ n2 ~< n3 ⇒ n1 ~< n3.

axiom orderStrict (n1,n2:message):
   n1 = n2 ⇒ n1 ~< n2 ⇒ false.

axiom orderSucc (n1,n2:message):
   n1 = n2 ⇒ n1 ~< Succ(n2).
```

Listing 1. Axioms in SQUIRREL.

This highlights an advantage of our method, which lets the user axiomatize any theory needed to conduct the security analysis. In contrast, symbolic approaches are less flexible, and only let the user provide axioms in some restricted ways.

To analyze the security of these two protocols, we proved some intermediate lemmas about state values, *e.g.* that counters are monotonically increasing. Our security analysis is similar to the one in GSVerif [15], and we obtain the same results, except that we provide computational guarantees (assuming that the MAC function used in CANAuth is EUF-CMA secure).

2) Protocols with stacks of hashes: We consider next a few RFID protocols which use a hash function to repeatedly refresh a secret value: the protocol of Example 1 and simplified versions of the SLK06 and YPLRK05 protocols of [24]. We prove authentication properties in all cases. These protocols are intended to be used with unkeyed hash functions, in the random oracle model (ROM), but we model them here with keyed hash functions (satisfying the PRF assumption) — a stronger assumption. For the SLK06 and YPLRK05 protocols we even assume that each tag uses its own hash key (shared with the reader) which rules out confusions between the hashes of different tags and thus simplifies proofs. For our running example, the proof of authentication is more complex, involving in particular a lemma showing that the chains of hash stacks of distinct agents remain disjoint, despite their use of the same hash key. For these protocols, it has sometimes proved useful to do some basic reasoning about updates of the memory cells, such as proving that the value of a state at a given timestamp is either equal to the initial value, or equal to the update term of the last action that updated this particular state; or proving that the value of a state remains constant between two consecutive updates.

3) Limitations: In order to faithfully model the ROM in our setting, it would be natural to provide the attacker with a hashing oracle, *i.e.* include in the protocol, for each hash function $\mathsf{H}(\_, \mathsf{k})$ in use, an action which upon input x outputs $\mathsf{H}(x, \mathsf{k})$. With this modeling, we can still use the EUF-CMA and PRF tactics associated to hash functions, but the exploration of possible occurrences of hashes now includes this additional oracle process. In order to prove an authentication property, we use the unforgeability of some hash function (rule EUF-CMA): if the reader receives a valid hash $\mathsf{H}(m, \mathsf{k})$, it must have been produced by some action of the protocol. In presence of an oracle, we would have to consider the possibility that the hash has been obtained by feeding m to the oracle. This possibility can however be dismissed because m is not known to the attacker. Proving such *weak secrecy* properties directly is difficult in our setting, but we can prove the strong secrecy of m (i.e. that it is indistinguishable from a fresh name) and conclude from there that it is also weakly secret. Unfortunately, the proof system of [18] does not give the possibility to derive reachability properties (here, weak secrecy) from equivalence properties (strong secrecy). Our extension of the proof system presented in the next section allows to lift this limitation.

V. Proof system

We now generalize the proof system of [18] in order to enable complex reasoning where reachability and equivalence properties are simultaneously established, and to improve automated reasoning.

A. Global meta-logic formulas

The *local* meta-logic formulas of Section III express properties that hold for all traces of a protocol. We now introduce *global* meta-logic formulas, which will make it possible to express properties involving multiple protocols, and will be the basis for our extended sequent calculus. In order to be able to simultaneously talk of the traces of multiple protocols, we need these protocols to be compatible.

Definition 6. *Two protocols are* compatible *if they use the same set $\mathcal{A}$ of action names and have the same partial order.*

The syntax of global meta-formulas is shown next, where we write α for a variable of any sort, including message, and $\vec{t}, \vec{t'}$ are sequences of meta-terms and meta-formulas.

$$\begin{aligned} F \ := \ & [\phi]_{\mathcal{P}} \mid [\vec{t} \sim \vec{t'}]_{\mathcal{P},\mathcal{P}'} \mid \tilde{\bot} \mid \tilde{\top} \mid \\ & \tilde{\forall}\alpha.F \mid \tilde{\exists}\alpha.F \mid F \mathrel{\tilde{\Rightarrow}} F' \mid F \mathrel{\tilde{\wedge}} F' \mid F \mathrel{\tilde{\vee}} F' \end{aligned}$$

To avoid the confusion with local meta-formulas we use other symbols for quantifiers and connectives. An atom of a global meta-formula may be a local formula relative to some protocol, or an equivalence relative to two protocols. We impose that all protocols used in a global meta-formula are compatible.

Similarly to local meta-formulas, 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 — notice the specific treatment of quantification over messages.

$$\begin{array}{ll} (\tilde{\bot})^{\mathbb{T}} \overset{\text{def}}{=} \bot & (\tilde{\forall}i.F)^{\mathbb{T}} \overset{\text{def}}{=} \wedge_{v \in \mathcal{D}_{\mathcal{I}}} (F)^{\mathbb{T}\{i \mapsto v\}} \\ ([\phi]_{\mathcal{P}})^{\mathbb{T}} \overset{\text{def}}{=} (\phi)^{\mathbb{T}}_{\mathcal{P}} \sim \mathsf{true} & (\tilde{\forall}\tau.F)^{\mathbb{T}} \overset{\text{def}}{=} \wedge_{v \in \mathcal{D}_{\mathcal{T}}} (F)^{\mathbb{T}\{\tau \mapsto v\}} \\ (F \mathrel{\tilde{\Rightarrow}} F')^{\mathbb{T}} \overset{\text{def}}{=} (F)^{\mathbb{T}} \Rightarrow (F')^{\mathbb{T}} & (\tilde{\forall}x.F)^{\mathbb{T}} \overset{\text{def}}{=} \forall x.(F)^{\mathbb{T}} \end{array}$$

$$([\vec{t} \sim \vec{t'}]_{\mathcal{P},\mathcal{P}'})^{\mathbb{T}} \overset{\text{def}}{=} (\vec{t})^{\mathbb{T}}_{\mathcal{P}} \sim (\vec{t'})^{\mathbb{T}}_{\mathcal{P}'}$$

The other cases are similarly handled.

Definition 7. *Two protocols $\mathcal{P}_1$ and $\mathcal{P}_2$ are* observationally equivalent *when they are compatible and the following global meta-logic formula is valid:*

$$\tilde{\forall}\tau.\ [\mathsf{happens}(\tau)]_{\mathcal{P}_1} \mathrel{\tilde{\Rightarrow}} [\mathsf{frame}@\tau \sim \mathsf{frame}@\tau]_{\mathcal{P}_1,\mathcal{P}_2}.$$

Note that $[\mathsf{happens}(\tau)]_{\mathcal{P}_1}$ and $[\mathsf{happens}(\tau)]_{\mathcal{P}_2}$ are equivalent as $\mathsf{happens}(\tau)$ only talks about the trace model.

B. Sequents

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

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

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.

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

$$\begin{array}{rcl} \Sigma;\, \Theta \vdash F & \rightsquigarrow & \tilde{\forall}\Sigma.(\Theta \mathrel{\tilde{\Rightarrow}} F) \\ \Sigma;\, \Theta : \Gamma \vdash_{\mathcal{P}} \phi & \rightsquigarrow & \tilde{\forall}\Sigma.(\Theta \mathrel{\tilde{\Rightarrow}} [\Gamma \Rightarrow \phi]_{\mathcal{P}}) \end{array}$$

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

Example 10. *To understand the distinction between Θ and Γ hypotheses in reachability sequents, note that the validity of $\Sigma;\, : \phi \vdash_{\mathcal{P}} \psi$ implies that of $\Sigma;\, [\phi]_{\mathcal{P}} : \vdash_{\mathcal{P}} \psi$, 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.*

When $\vec{u}$ and $\vec{v}$ are sequences of meta-terms or meta-formulas of the same length, we write $\Sigma;\, \Theta \vdash_{\mathcal{P},\mathcal{P}'} \vec{u} \sim \vec{v}$ for $\Sigma;\, \Theta \vdash [\vec{u} \sim \vec{v}]_{\mathcal{P},\mathcal{P}'}$.

C. Structural rules

Reachability sequents (resp. general sequents) generalize the reachability (resp. equivalence) sequents of [18], and the proof rules for that earlier work can naturally be adapted for our generalized sequents. This initial set of proof rules includes, for both kinds of sequents, the usual first-order reasoning rules and proof by induction for both kinds of sequents but also rules that embody cryptographic assumptions or reasoning about names. We shall not comment further these (essentially) unchanged rules, but refer the reader to [18] for more details.

The semantics of local meta-formulas is not boolean, but probabilistic: a formula is valid if its meta-interpretation is true with overwhelming probability. It is thus remarkable that the usual rules of first-order logic are sound for these formulas. But this only holds when we remain within the "local" part of

$$\frac{\begin{array}{c}\Sigma;\,\Theta:\,\Gamma,\phi\vdash_{\mathcal{P}}\psi\\ \Sigma;\,\Theta:\,\Gamma,\phi'\vdash_{\mathcal{P}}\psi\end{array}}{\Sigma;\,\Theta:\,\Gamma,\phi\vee\phi'\vdash_{\mathcal{P}}\psi}\qquad\frac{\begin{array}{c}\Sigma;\,\Theta,[\phi]_{\mathcal{P}}:\,\Gamma\vdash_{\mathcal{P}}\psi\\ \Sigma;\,\Theta,[\phi']_{\mathcal{P}}:\,\Gamma\vdash_{\mathcal{P}}\psi\end{array}}{\Sigma;\,\Theta,[\phi\vee\phi']_{\mathcal{P}}:\,\Gamma\vdash_{\mathcal{P}}\psi}$$

In the right rule, we require that one of the two disjuncts is a pure trace formula.

Fig. 3. Left disjunction rules at the local and global levels.

Global-Local

$$\frac{\Sigma;\,\Theta\vdash[\phi]_{\mathcal{P}}}{\Sigma;\,\Theta:\,\vdash_{\mathcal{P}}\phi}$$

Local-Global

$$\frac{\Sigma;\,\Theta:\,\vdash_{\mathcal{P}}\phi}{\Sigma;\,\Theta\vdash[\phi]_{\mathcal{P}}}$$

Equiv-Term

$$\frac{\begin{array}{c}\Sigma;\,\Theta:\,\vdash_{\mathcal{P}}t=t'\\ \Sigma;\,\Theta\vdash_{\mathcal{P},\mathcal{P}'}\vec{C}[t']\sim\vec{v}\end{array}}{\Sigma;\,\Theta\vdash_{\mathcal{P},\mathcal{P}'}\vec{C}[t]\sim\vec{v}}$$

Equiv-Form

$$\frac{\begin{array}{c}\Sigma;\,\Theta:\,\vdash_{\mathcal{P}}\phi\Leftrightarrow\psi\\ \Sigma;\,\Theta\vdash_{\mathcal{P},\mathcal{P}'}\vec{C}[\psi]\sim\vec{v}\end{array}}{\Sigma;\,\Theta\vdash_{\mathcal{P},\mathcal{P}'}\vec{C}[\phi]\sim\vec{v}}$$

Rewrite-Equiv

$$\frac{\Sigma;\,\Theta\vdash_{\mathcal{P},\mathcal{P}'}(\Gamma\Rightarrow\phi)\sim(\Delta\Rightarrow\psi)\qquad\Sigma;\,\Theta:\,\Delta\vdash_{\mathcal{P}'}\psi}{\Sigma;\,\Theta:\,\Gamma\vdash_{\mathcal{P}}\phi}$$

Fig. 4. Inference rules involving mixed kinds of sequents.

local sequents, as illustrated in Figure 3. The second rule in that figure would not be valid without the proviso that one of the disjuncts is a pure trace formula. A pure trace formula is a local meta-formula which does not feature any macros, and is always true or false when evaluated in a trace model.

Definition 8. *A local meta-formula ϕ is a* pure trace formula *w.r.t. protocol $\mathcal{P}$ if ϕ does not contain any macro and if for any trace model $\mathbb{T}$, $(\phi)^{\mathbb{T}}_{\mathcal{P}}\sim\mathsf{true}$ or $(\phi)^{\mathbb{T}}_{\mathcal{P}}\sim\mathsf{false}$ is valid.*

A simple syntactic criterion guaranteeing this property is that the formula does not feature any macro or message, *i.e.* its atoms are only comparisons of indices and timestamps.

Essentially, assuming that $\phi\vee\phi'$ is true with overwhelming probability, we cannot generally conclude that one of ϕ and ϕ' is also true with overwhelming probability; the pure trace formula proviso allows one to fix this gap because a pure trace formula is either always true or always false (depending on the trace model). Proofs in our case studies must sometimes be carefully structured to be able to accommodate this subtlety.

Our proof system also features rules which articulate the relationship between our two kinds of sequents, shown in Figure 4. Rules Global-Local and Local-Global are obvious conversions. Rules Equiv-Term and Equiv-Form are immediate generalizations of the identically named rules in [18], with the new environment Θ allowing to carry hypotheses from the equivalence sequent in conclusion to the reachability sequent in first premise. Conversely, the new rule Rewrite-Equiv shows how an equivalence judgement can be used to help derive reachability judgements. As expected all these rules have been shown to be sound.

Proposition 1. *The rules in Figures 3 and 4 are sound.*

Example 11. *To illustrate the utility of this articulation between equivalence and reachability, we show how to prove a (weak) secrecy property (expressed as a reachability sequent) using a strong secrecy property (an equivalence) as hypothesis. Consider a protocol $\mathcal{P}$ involving a memory cell s of index arity 1. Given a fresh name m, we consider the reachability sequent $\Sigma;\,\Theta_{\mathsf{hap}},\Theta:\,\vdash_{\mathcal{P}}\phi$ where $\Sigma\stackrel{def}{=}\tau,\tau',i$ and:*

$$\begin{aligned}\Theta_{\mathsf{hap}}&\stackrel{def}{=}[\mathsf{happens}(\mathsf{pred}(\tau))]_{\mathcal{P}}\\ \Theta&\stackrel{def}{=}[\mathsf{frame}@\tau,\mathsf{s}[i]@\tau'\sim\mathsf{frame}@\tau,\mathsf{m}]_{\mathcal{P},\mathcal{P}}\\ \phi&\stackrel{def}{=}\mathsf{input}@\tau\neq\mathsf{s}[i]@\tau'\end{aligned}$$

This sequent states that, if the values stored in the memory cell s are strongly secret (this is the global meta-formula Θ), then we know that $\mathsf{s}[i]@\tau'$ is (weakly) secret, i.e. *the attacker cannot deduce its value (this is the local meta-formula ϕ). In order to derive this sequent, we use the rule* Rewrite-Equiv *with $\psi:=\mathsf{input}@\tau\neq\mathsf{m}$ and empty environments Γ and Δ. We thus have to derive the following two sequents.*

$$\Sigma;\,\Theta_{\mathsf{hap}},\Theta\vdash_{\mathcal{P},\mathcal{P}}\mathsf{input}@\tau\neq\mathsf{s}[i]@\tau'\sim\mathsf{input}@\tau\neq\mathsf{m}$$
$$\Sigma;\,\Theta_{\mathsf{hap}},\Theta:\,\vdash_{\mathcal{P}}\mathsf{input}@\tau\neq\mathsf{m}$$

The second one can be easily derived using the freshness of the name m. Concerning the first one, we shall see that we can use the equivalence in Θ to deduce the right side of the sequent: this is the purpose of the next subsection. The file corresponding to this example in [16] is `running-ex-secrecy.sp`.

Echoing the limitations discussed at the end of Section IV, the articulation illustrated above is also critical to prove the strong secrecy of the states of tags in a variant of the OSK protocol with oracles but no readers. Interestingly, strong and weak secrecy must be derived simultaneously in that proof, in a mutual induction. See the file `running-ex-oracle.sp` at [16] for more details.

D. Automating proofs by bi-deduction

In the sequent $\Sigma;\,\Theta_{\mathsf{hap}},\Theta\vdash_{\mathcal{P},\mathcal{P}}\phi\sim\psi$ of Example 11, the left side of the equivalence in Θ contains all the necessary information to compute ϕ: there exists a machine $\mathcal{B}$ computing the latter from the former. Moreover, the *same* algorithm $\mathcal{B}$ computes ψ from the right side of the equivalence in Θ. Intuitively, the proof is done here: if you can break the equivalence $\phi\sim\psi$, then you can break the equivalence Θ by composing the distinguisher for $\phi\sim\psi$ with $\mathcal{B}$. The existence of such a $\mathcal{B}$ proves that an equivalence is entailed by another equivalence. Informally:

$$\frac{\exists\mathcal{B}\text{ s.t. }\mathcal{B}\text{ computes }\vec{v}_i\text{ from }\vec{u}_i\text{ w.r.t. }\mathcal{P}_i\text{ for any }i\in\{1,2\}}{\Sigma;\,\Theta,[\vec{u}_1\sim\vec{u}_2]_{\mathcal{P}_1,\mathcal{P}_2}\vdash[\vec{v}_1\sim\vec{v}_2]_{\mathcal{P}_1,\mathcal{P}_2}}\tag{1}$$

We shall now define formally this notion of simultaneous computation, which we call *bi-deduction* and note $\Sigma;\,\Theta;\,\#(\vec{u}_1,\vec{u}_2)\rhd_{\mathcal{P}_1,\mathcal{P}_2}\#(\vec{v}_1,\vec{v}_2)$.

1) Bi-terms: A bi-term $t_\#$ is an element of the form $\#(t_1, t_2)$ where t_1 and t_2 are meta-terms. Similarly, a bi-formula $\#(\phi_1, \phi_2)$ is a pair of local formulas. We often use compact notations to refer to bi-terms (and bi-formulas) sharing some top-level constructs.

For example, $f(\#(t_1, t_2), g(\#(s_1, s_2)))$ is the bi-term $\#(f(t_1, g(s_1)), f(t_2, g(s_2)))$, and $\mathsf{seq}(\vec{\alpha} : \#(t_1, t_2))$ is the bi-term $\#(\mathsf{seq}(\vec{\alpha} : t_1), \mathsf{seq}(\vec{\alpha} : t_2))$. We may also note t for the bi-term $\#(t, t)$. We similarly lift boolean connectives to bi-terms: *e.g.* $\#(\phi_1, \phi_2) \wedge \#(\psi_1, \psi_2)$ is $\#(\phi_1 \wedge \psi_1, \phi_2 \wedge \psi_2)$.

2) Bi-deduction: A bi-deduction judgement is an element of the form $\Sigma;\ \Theta;\ \vec{u}_\# \triangleright_{\mathcal{P}_1,\mathcal{P}_2} \{v_\# \mid \phi_\#\}$ where Σ is a sequence of variables of any sort, Θ is a set of global meta-formulas, $\mathcal{P}_1, \mathcal{P}_2$ are protocols, $\vec{u}_\#$ is a sequence of bi-terms, v is a bi-term, and $\phi_\#$ is a bi-formula. We require that Σ binds all variables in the judgement. We omit the protocols $\mathcal{P}_1, \mathcal{P}_2$ from the judgement when they are clear from the context, and write $\Sigma;\ \Theta;\ \vec{u}_\# \triangleright v_\#$ when $\phi_\# = \top$. We also write $\Sigma;\ \Theta;\ \vec{u}_\# \triangleright \vec{v}_\#$ when $\Sigma;\ \Theta;\ \vec{u}_\# \triangleright v^i_\#$ for each element $v^i_\#$ in $\vec{v}_\#$.

A judgement $\Sigma;\ \Theta;\ \#(\vec{u_1}, \vec{u_2}) \triangleright_{\mathcal{P}_1,\mathcal{P}_2} \{\#(v_1, v_2) \mid \#(\phi_1, \phi_2)\}$ is valid when, for any trace model $\mathbb{T}$, there exist probabilistic PTIME machines $\mathcal{B}_o$ and $\mathcal{B}_c$ such that, for any computational model $\mathbb{M}$ and valuation σ such that $\mathbb{M}, \sigma \models (\Theta)^{\mathbb{T}}$, and for any $i \in \{1, 2\}$, the following equalities hold with overwhelming probability:

$$\begin{aligned} &[\![\mathsf{EQ}]\!]_{\mathbb{M}}\big(\mathcal{B}_c([\![(\vec{u}_i)^{\mathbb{T}}_{\mathcal{P}_i}]\!]^{\sigma}_{\mathbb{M}}),\ [\![(\phi_i)^{\mathbb{T}}_{\mathcal{P}_i}]\!]^{\sigma}_{\mathbb{M}}\big) = 1 \\ &[\![\mathsf{EQ}]\!]_{\mathbb{M}}\big(\mathcal{B}_o([\![(\vec{u}_i)^{\mathbb{T}}_{\mathcal{P}_i}]\!]^{\sigma}_{\mathbb{M}}),\ [\![(\text{if } \phi_i \text{ then } v_i)^{\mathbb{T}}_{\mathcal{P}_i}]\!]^{\sigma}_{\mathbb{M}})\big) = 1 \end{aligned} \tag{2}$$

Said otherwise, given $\vec{u}_i$ as input, the machines $\mathcal{B}_c$ and $\mathcal{B}_o$ respectively compute ϕ_i and (if ϕ_i then v_i) (up to a negligible probability of failure) whenever Θ holds, for $i = 1$ and protocol $\mathcal{P}_1$, and for $i = 2$ and protocol $\mathcal{P}_2$. Note that the *same* machines are used on *both* sides.

Example 12. *Let i', i be index variables. The bi-deduction judgement $i', i;\ ;\ \triangleright i' = i$ is valid because, for any trace model $\mathbb{T}$, we can choose $\mathcal{B}$ to be the machine that always returns true if $(i' = i)^{\mathbb{T}}$ is true, and false otherwise. The dependence of $\mathcal{B}$ on $\mathbb{T}$ is crucial here.*

Example 13. *Let $\mathsf{a}[\_]$ be some action name of index arity one, τ a timestamp variable and i, j index variables. The bi-deduction judgement:*

$$\tau, j;\ ;\ \mathsf{seq}(i\colon \mathsf{n}[i] \mid \mathsf{a}[i] \leq \tau) \triangleright \{\mathsf{n}[j] \mid \mathsf{a}[j] < \tau\}$$

is valid because the sequence $\mathsf{seq}(i\colon \mathsf{n}[i] \mid \mathsf{a}[i] \leq \tau)$ contains the names $\mathsf{n}[i]$ for any $\mathsf{a}[i] \leq \tau$, and consequently contains $\mathsf{n}[j]$ whenever $\mathsf{a}[j] < \tau$. Hence we can easily construct a machine $\mathcal{B}$ extracting $\mathsf{n}[j]$ from the sequence.

3) Rules: We can now give a formal version of Equ. (1), *i.e.* state the rule that derives equivalences from bi-deductions:

$$\textsc{Bi-Deduce}\quad \frac{\Sigma;\ \Theta;\ \#(\vec{u}_1, \vec{u}_2) \triangleright_{\mathcal{P}_1,\mathcal{P}_2} \#(\vec{v}_1, \vec{v}_2)}{\Sigma;\ \Theta, [\vec{u}_1 \sim \vec{u}_2]_{\mathcal{P}_1,\mathcal{P}_2} \vdash [\vec{v}_1 \sim \vec{v}_2]_{\mathcal{P}_1,\mathcal{P}_2}}$$

If, given some known bi-terms $\#(\vec{u}_1, \vec{u}_2)$, we can bi-deduce some other bi-terms $\#(\vec{v}_1, \vec{v}_2)$ (for a pair of protocols $\mathcal{P}_1, \mathcal{P}_2$), then we know that the equivalence $[\vec{u}_1 \sim \vec{u}_2]_{\mathcal{P}_1,\mathcal{P}_2}$ entails $[\vec{v}_1 \sim \vec{v}_2]_{\mathcal{P}_1,\mathcal{P}_2}$.

Example 14. *Let us come back to Example 11 in which we had to derive the following sequent:*

$$\begin{aligned} \Sigma;\ \Theta_{\mathsf{hap}}, [&\mathsf{frame}@\tau, \mathsf{s}[i]@\tau' \sim \mathsf{frame}@\tau, \mathsf{m}]_{\mathcal{P},\mathcal{P}} \\ &\vdash_{\mathcal{P},\mathcal{P}} \mathsf{input}@\tau \neq \mathsf{s}[i]@\tau' \sim \mathsf{input}@\tau \neq \mathsf{m} \end{aligned}$$

Using the rule Bi-Deduce*, we are left with the following bi-deduction judgement:*

$$\Sigma;\ \Theta_{\mathsf{hap}};\ \mathsf{frame}@\tau, \#(\mathsf{s}[i]@\tau', \mathsf{m}) \triangleright_{\mathcal{P},\mathcal{P}} \mathsf{input}@\tau \neq \#(\mathsf{s}[i]@\tau', \mathsf{m})$$

That judgement is valid according to our semantics: indeed there is a machine that can (1) obtain $x := \mathsf{frame}@\tau$ and $y := \mathsf{s}[i]@\tau'$ when $i = 1$ (resp. $y := \mathsf{m}$ when $i = 2$); (2) build $z := \mathsf{input}@\tau$ from $\mathsf{frame}@\tau$; (3) compute $z \neq y$.

We present in Figure 5 some rules for bi-deducibility; a complete presentation is given in Figure 8, Appendix A. Rule SEQ-DED allows to bi-deduce a bi-term $\#(v_1, v_2)$, whenever $\#(\phi_1, \phi_2)$ holds, from a sequence $\mathsf{seq}(\vec{\alpha} : \#(u_1, u_2) \mid \#(\psi_1, \psi_2))$ if it appears in the sequence *simultaneously* on both sides: there must exists $\vec{\alpha}$ such that, for any $i \in \{1, 2\}$, $u_i = v_i$ and the sequence condition ψ_i holds whenever ϕ_i holds. Rule PURE-DED states that pure trace formulas are always bi-deducible: indeed, once the trace model is fixed, they have a constant value (either always true or always false). Rule PAIR-PROJ allows to exploit the fact that a machine can recover a pair's components. The FRAME rule states that $\mathsf{frame}@T$ contains all $\mathsf{frame}@T'$ for any $T' \leq T$. Combined with other rules, this makes it possible to bi-deduce any past input from a frame. The rule FA says that the image of a function $f(\vec{v}_\#)$ is bi-deducible whenever its arguments $\vec{v}_\#$ are bi-deducible. In case f is of arity zero, we must also check that the condition $\phi_\#$ is bi-deducible — see rule FA_0. Finally, FA-ITE allows one to derive a bi-deduction with a conditional on its right, taking advantage of its condition to refine the premises: we know that we must deduce the then branch only if the condition ψ holds, and similarly for the else branch.

Proposition 2. *The* Bi-Deduce *rule and the rules in Figure 5 (as well as the complete version of the rules given in Figure 8 in Appendix) are sound.*

Example 15. *We now formally derive the bi-deduction judgement of Example 14. We first apply* FA *on the $\neq$ function symbol. This gives us two bi-deduction judgements:*

1) *$\Sigma;\ \Theta_{\mathsf{hap}};\ \mathsf{frame}@\tau, \#(\mathsf{s}[i]@\tau', \mathsf{m}) \triangleright_{\mathcal{P},\mathcal{P}} \mathsf{input}@\tau$; and*
2) *$\Sigma;\ \Theta_{\mathsf{hap}};\ \mathsf{frame}@\tau, \#(\mathsf{s}[i]@\tau', \mathsf{m}) \triangleright_{\mathcal{P},\mathcal{P}} \#(\mathsf{s}[i]@\tau', \mathsf{m})$.*

The second judgement is derived using SEQ-DED*, since the bi-term we must deduce is already present in the left side of the judgement. For the first one, we replace $\mathsf{input}@\tau$ by $\mathsf{att}(\mathsf{frame}@\mathsf{pred}(\tau))$: this is done by unfolding the* input *macro (rule not shown). Then, we get rid of the* att *function symbol using* FA*. Applying* FRAME *then gives:*

SEQ-DED

$$\frac{\begin{array}{c}\Sigma;\ \Theta \vdash \tilde{\exists}\vec{\alpha}.\ \begin{pmatrix}[\phi_1 \Rightarrow (\psi_1 \wedge u_1 = v_1)]_{\mathcal{P}_1} \tilde{\wedge} \\ [\phi_2 \Rightarrow (\psi_2 \wedge u_2 = v_2)]_{\mathcal{P}_2}\end{pmatrix} \\ \Sigma;\ \Theta;\ \vec{u}_{\#}, \mathsf{seq}(\vec{\alpha} : \#(u_1, u_2) \mid \#(\psi_1, \psi_2)) \triangleright \#(\phi_1, \phi_2)\end{array}}{\Sigma;\ \Theta;\ \vec{u}_{\#}, \mathsf{seq}(\vec{\alpha} : \#(u_1, u_2) \mid \#(\psi_1, \psi_2)) \triangleright \{\#(v_1, v_2) \mid \#(\phi_1, \phi_2)\}}$$

PURE-DED

$$\frac{v, \phi \text{ pure trace formula}}{\Sigma;\ \Theta;\ \vec{u}_{\#} \triangleright \{\#(v, v) \mid \#(\phi, \phi)\}}$$

PAIR-PROJ

$$\frac{\Sigma;\ \Theta;\ \vec{u}_{\#}, \mathsf{seq}(\vec{\alpha} : u^1_{\#} \mid \psi_{\#}), \mathsf{seq}(\vec{\alpha} : u^2_{\#} \mid \psi_{\#}) \triangleright \{v_{\#} \mid \phi_{\#}\}}{\Sigma;\ \Theta;\ \vec{u}_{\#}, \mathsf{seq}(\vec{\alpha} : \langle u^1_{\#}, u^2_{\#}\rangle \mid \psi_{\#}) \triangleright \{v_{\#} \mid \phi_{\#}\}}$$

FRAME

$$\frac{\Sigma;\ \Theta;\ \vec{u}_{\#}, \mathsf{seq}(\vec{\alpha}, T' : \mathsf{frame}@T' \mid \psi_{\#} \wedge T' \leq T) \triangleright \{v_{\#} \mid \phi_{\#}\}}{\Sigma;\ \Theta;\ \vec{u}_{\#}, \mathsf{seq}(\vec{\alpha} : \mathsf{frame}@T \mid \psi_{\#}) \triangleright \{v_{\#} \mid \phi_{\#}\}}$$

FA

$$\frac{\Sigma;\ \Theta;\ \vec{u}_{\#} \triangleright \{\vec{v}_{\#} \mid \phi_{\#}\}}{\Sigma;\ \Theta;\ \vec{u}_{\#} \triangleright \{f(\vec{v}_{\#}) \mid \phi_{\#}\}}$$

FA$_0$

$$\frac{\Sigma;\ \Theta;\ \vec{u}_{\#} \triangleright \phi_{\#}}{\Sigma;\ \Theta;\ \vec{u}_{\#} \triangleright \{f() \mid \phi_{\#}\}}$$

FA-ITE

$$\frac{\begin{array}{c}\Sigma;\ \Theta;\ \vec{u}_{\#} \triangleright \{\psi_{\#} \mid \phi_{\#}\} \\ \Sigma;\ \Theta;\ \vec{u}_{\#} \triangleright \{t_{\#} \mid \phi_{\#} \wedge \psi_{\#}\} \\ \Sigma;\ \Theta;\ \vec{u}_{\#} \triangleright \{t'_{\#} \mid \phi_{\#} \wedge \neg\psi_{\#}\}\end{array}}{\Sigma;\ \Theta;\ \vec{u}_{\#} \triangleright \{\mathsf{if}\ \psi_{\#}\ \mathsf{then}\ t_{\#}\ \mathsf{else}\ t'_{\#} \mid \phi_{\#}\}}$$

We identify a single term u with the degenerate sequence $\mathsf{seq}(\epsilon\colon u \mid \top)$ so that the left rules apply not only to sequences. The FA rule only applies when f is of non-zero arity.

Fig. 5. Bi-deduction rules for protocols $\mathcal{P}_1, \mathcal{P}_2$.

$$\Sigma;\ \Theta_{\mathsf{hap}};\ \mathsf{seq}(T\colon \mathsf{frame}@T \mid T \leq \tau) \triangleright_{\mathcal{P},\mathcal{P}} \mathsf{frame}@\mathsf{pred}(\tau).$$

We conclude with the rule SEQ-DED, *noticing that we can instantiate T by* $\mathsf{pred}(\tau)$.

Using our bi-deduction rules, we implemented a procedure $\mathsf{DED}(\cdot)$ which automatically checks whether a bi-deduction judgement $\Sigma;\ \Theta;\ \vec{u}_{\#} \triangleright \{v_{\#} \mid \phi_{\#}\}$ is valid. Our procedure is sound but incomplete: it relies on heuristics to decide on which order it applies the rules. Studying the decidability and complexity of the bi-deduction logic is left as future work.

This procedure has been incorporated in SQUIRREL `apply` tactic: when applying a lemma `H` whose conclusion is an equivalence, we use our automatic procedure to check that the current goal is bi-deducible from `H`'s conclusion. This improvement plays a crucial role to ease the proof effort for the case studies presented in this paper, but has also allowed to simplify older case studies from [18].

a) Discussion: Some bi-deduction proofs can be done directly using equivalence judgements and existing rules. Roughly, the idea is to construct an explicit context computing the wanted terms $\vec{v}$ from the known terms $\vec{u}$, using DUP, FA.

But this is not always possible, in particular when sequences are involved. E.g., using BI-DEDUCE, it is possible to prove (the two sides are syntactically equal but interpreted w.r.t. distinct protocols):

$$\tau;\ \begin{bmatrix}\mathsf{frame}@\tau \sim \\ \mathsf{frame}@\tau\end{bmatrix}_{\mathcal{P}_1,\mathcal{P}_2} \vdash \begin{bmatrix}\mathsf{seq}(\tau_1\colon \mathsf{input}@\tau_1 \mid \tau_1 < \tau) \sim \\ \mathsf{seq}(\tau_1\colon \mathsf{input}@\tau_1 \mid \tau_1 < \tau)\end{bmatrix}_{\mathcal{P}_1,\mathcal{P}_2}$$

using the fact that $\mathsf{frame}@\tau$ contains $\mathsf{frame}@\tau_0$ for all $\tau_0 \leq \tau$, and the definition of input — see the long version [20] for a detailed proof. With our current set of rules, we believe this judgement cannot be proved without BI-DEDUCE, as we cannot construct an explicit context extracting $\mathsf{frame}@\tau_0$ from $\mathsf{frame}@\tau$ for $\tau_0 \leq \tau$. Indeed, such a context would have to perform projections on frames a number of times which depends on the trace models, which cannot be done.

E. Induction and bi-deduction

The automatic bi-deduction procedure derived from the bi-deduction proof system presented so far is very efficient and helpful. In some cases, though, it is not enough to conclude, at least not with a reasonable proof effort.

Example 16. *Consider the protocol of our running example, with tags only for simplicity. Call it $\mathcal{P}_1$, and let $\mathcal{P}_2$ be a modified version where the states $\mathsf{s}_{\mathsf{T}}[i]$ are initialized with $\mathsf{n}^0[i]$ instead of $\mathsf{s}^0[i]$. Consider the equivalence*

$$\begin{array}{l}\mathsf{frame}@t, \mathsf{k}, \mathsf{seq}(i\colon \mathsf{s}^0[i]) \sim_{\mathcal{P}_1,\mathcal{P}_2} \\ \mathsf{frame}@t, \mathsf{k}, \mathsf{seq}(i\colon \mathsf{n}^0[i])\end{array} \tag{3}$$

where we reveal, in addition to the frames, the key k and the initial values for all i. This equivalence obviously holds because it compares terms that only differ by a bijective renaming. Consider now an even stronger equivalence where we reveal all past states:

$$\begin{array}{l}\mathsf{frame}@\tau, \mathsf{k}, \mathsf{seq}(i, \tau'\colon \mathsf{if}\ \tau' \leq \tau\ \mathsf{then}\ \mathsf{s}_{\mathsf{T}}[i]@\tau') \sim_{\mathcal{P}_1,\mathcal{P}_2} \\ \mathsf{frame}@\tau, \mathsf{k}, \mathsf{seq}(i, \tau'\colon \mathsf{if}\ \tau' \leq \tau\ \mathsf{then}\ \mathsf{s}_{\mathsf{T}}[i]@\tau')\end{array} \tag{4}$$

Such enriched equivalences are typically needed for use with REWRITE-EQUIV, *as is the case in our YubiHSM case study. In our simple example, the enriched equivalence (4) obviously holds because it is a renaming, but it is difficult to derive it without an ad-hoc renaming rule, which would be inapplicable in realistic cases such as our YubiHSM case study. Intuitively, equivalence (4) follows from (3) because the attacker can bi-deduce the successive states from their initial values and k. However, the number of steps of the construction of $\mathsf{s}_{\mathsf{T}}[i]@\tau'$ depends on the position of τ' in a trace model, so we need some sort of inductive reasoning to establish this bi-deduction.*

At a high-level, this inductive reasoning would proceed as follows. If τ is init, *the sequence of states is composed of initial values only, which are given in the first equivalence. If τ is* undef, *the comparison $\tau' \leq \tau$ cannot hold hence the sequence* *is essentially empty. Otherwise, τ is of the form $\mathsf{T}[i', j]$. By induction hypothesis we have already bi-deduced the part of the sequence corresponding to values of τ' such that $\tau' < \tau$. It remains to bi-deduce the states for $\tau' = \tau = \mathsf{T}[i', j]$. But, for any i, we have (for both $\mathcal{P}_1$ and $\mathcal{P}_2$)*

$$\begin{aligned}\mathsf{s_T}[i]@\mathsf{T}[i', j] \overset{\textit{def}}{=} \mathsf{if}\ i = i'\ \mathsf{then}\ &\mathsf{H}(\underline{\mathsf{s_T}[i']@\mathsf{pred}(\mathsf{T}[i', j])}, \underline{\mathsf{k}})\\ \mathsf{else}\ &\underline{\mathsf{s_T}[i]@\mathsf{pred}(\mathsf{T}[i', j])}\end{aligned}$$

which can be bi-deduced, notably because the parts underlined above have already been shown to be bi-deducible.

The reasoning performed in the previous example cannot be formally carried out with the bi-deduction inference rules presented so far, because they do not include any form of inductive reasoning. Of course, an induction and case analysis over τ could be done at the general sequent level, but this would require the user to supply the inductive invariant and carry out a very tedious manual proof, as shown in `running-ex-deduction.sp` on the previous example.

Instead, we extend our automated procedure for bi-deduction with inductive reasoning. The soundness of our procedure relies on the following induction rule:

$$\textsc{Ind-}\triangleright\quad \frac{\Sigma, \tau;\ \Theta;\ \vec{u}_\#, \mathsf{seq}(\tau' \colon \vec{v}_\#[\tau'] \mid \phi_\# \wedge \tau' < \tau) \triangleright \{\vec{v}_\#[\tau] \mid \phi_\#\}}{\Sigma;\ \Theta;\ \vec{u}_\# \triangleright \{\mathsf{seq}(\tau \colon \vec{v}_\#[\tau]) \mid \phi_\#\}}$$

Assume we want to prove the bi-deduction judgement $\Sigma;\ \Theta;\ u_\# \triangleright v_\#$. First, our procedure computes an invariant of the form $U_\# \overset{\text{def}}{=} \lambda\tau, u_\#[\tau]$ such that $U_\#$ is an invariant of the protocol:

$$\Sigma, \tau;\ \Theta;\ \vec{u}_\#, \mathsf{seq}(\tau' \colon U_\#[\tau'] \mid \tau' < \tau) \triangleright U_\#[\tau]$$

This invariant $U_\#$ is computed automatically, using abstract interpretation techniques [25]. Finally, we check whether

$$\Sigma;\ \Theta;\ u_\#, \mathsf{seq}(\tau \colon U_\#[\tau]) \triangleright v_\#$$

using our automated procedure $\mathsf{DED}(\cdot)$. We refer the reader to Appendix B for the proof of soundness of this approach and additional details on how the invariant $U_\#$ is computed.

This procedure for bi-deduction extended with inductive reasoning has been incorporated in the SQUIRREL `apply` tactic. Because it is more costly, it is not used by default, and must be turned on by using `apply ~inductive`.

Example 17. *The equivalence in Equ.* (4) *of Example 16 is proved directly using our bi-deduction procedure with inductive reasoning (see* `running-ex-deduction.sp` *in [16]).*

F. Sequences

We often use sequences when reasoning on protocols. *e.g.* to enrich equivalences with collections of messages that are already known to the attacker or can safely be disclosed as far as the property being proved is concerned. We give in Figure 6 two new rules that are sometimes necessary to prove equivalences between sequences.

$$\textsc{Split-Seq}\quad \frac{\Sigma;\ \Theta \vdash_{\mathcal{P},\mathcal{P}'} \begin{array}{c}\vec{u}, \mathsf{seq}(\vec{\alpha} : \mathsf{if}\ \phi\ \mathsf{then}\ t), \mathsf{seq}(\vec{\alpha} : \mathsf{if}\ \neg\phi\ \mathsf{then}\ t)\\ \sim \vec{v}, \mathsf{seq}(\vec{\alpha} : \mathsf{if}\ \psi\ \mathsf{then}\ s), \mathsf{seq}(\vec{\alpha} : \mathsf{if}\ \neg\psi\ \mathsf{then}\ s)\end{array}}{\Sigma;\ \Theta \vdash_{\mathcal{P},\mathcal{P}'} \vec{u}, \mathsf{seq}(\vec{\alpha} : t) \sim \vec{v}, \mathsf{seq}(\vec{\alpha} : s)}$$

$$\textsc{Const-Seq}\quad \frac{\begin{array}{c}\Sigma;\ \Theta \vdash \left[\forall\vec{\alpha}, \bigvee_{0\le i\le n} b_i\right]_{\mathcal{P}}\\ \Sigma;\ \Theta \vdash \left[\bigwedge_{0\le i\le n} \forall\vec{\alpha}, b_i \Rightarrow t = t_i\right]_{\mathcal{P}}\\ \Sigma;\ \Theta \vdash \left[\bigwedge_{0\le i\le n} \forall\vec{\alpha}, b_i \Rightarrow s = s_i\right]_{\mathcal{P}'}\\ \vec{\alpha} \cap \Sigma = \emptyset \qquad (b_i)_{1\le i\le n}\ \text{pure trace formulas}\\ \Sigma;\ \Theta \vdash_{\mathcal{P},\mathcal{P}'} \vec{u}, t_1, \ldots, t_n \sim \vec{v}, s_1, \ldots, s_n\end{array}}{\Sigma;\ \Theta \vdash_{\mathcal{P},\mathcal{P}'} \vec{u}, \mathsf{seq}(\vec{\alpha} : t) \sim \vec{v}, \mathsf{seq}(\vec{\alpha} : s)}$$

Fig. 6. Inference rules for equivalence of sequences.

The SPLIT-SEQ rule allow to split sequences in two, according to some arbitrary meta-formula ϕ on the left and ψ on the right. The CONST-SEQ rule can be used to simplify a sequence $\mathsf{seq}(\vec{\alpha} : t)$ into a finite collection $t_1, \ldots, t_n$ when it can be proved that all instances of t fall into this finite set of possibilities. Since we are dealing with an equivalence with sequences on both sides, this simplification must be done simultaneously on both sides, and we must make sure that the mapping for the instances of t to the constants t_i is the same as its counterpart on the other side of the equivalence: this is the role of the conditions b_i that partition the sequences. As expected, these rules are sound.

Proposition 3. *The rules in Figure 6 are sound.*

VI. CASE STUDY: YUBIKEY AND YUBIHSM

This section discusses the YubiKey and the YubiHSM protocols following the security analysis done in [19], [26] using the symbolic prover TAMARIN. The analysis of YubiHSM in TAMARIN requires the use of the interactive mode, and took the authors around one month. Doing this analysis in SQUIRREL allows to test our extensions of the tool and to provide stronger computational guarantees. Conducting this analysis in SQUIRREL required a similar amount of human effort than the TAMARIN analysis. The corresponding SQUIRREL files, `yubikey.sp` (270 LoC) and `yubihsm.sp` (720 LoC), can be found at [16].

A. Background

The YubiKey is a simple physical authentication device with a unique button. This device, manufactured by Yubico, allows users to securely authenticate to their accounts by issuing a one-time password (OTP). In its typical configuration, the YubiKey generates a random OTP by encrypting a secret value and a counter. This message is accepted by the server only if it decrypts under the correct key to a valid secret value containing a counter whose value is larger than the last value accepted by the server for that YubiKey.

We first analyze the security of the protocol assuming that the server remains secure. Then, we consider an adversary having access to the authentication server. To provide some security guarantees in this scenario, a Hardware Security Module (HSM), called YubiHSM, is used to protect the working secret symmetric keys.

As in [19], [26], instead of having a session counter and a token counter, we consider a single counter. To conduct the security analysis in TAMARIN, the authors in [19], [26] over-approximate the behaviours of the system by allowing the adversary to instantiate the rule for any counter value that is higher than the previous one. We do not rely on this approximation in our SQUIRREL security analysis.

B. *YubiKey*

Yubico assigns a key k, as well as a public and a secret identifier (*pid*, *sid*) to each YubiKey. The counter inside the YubiKey is incremented whenever the YubiKey is plugged in, as well as when an OTP is generated, *i.e.* when the button of the YubiKey is pressed. This OTP is obtained by encrypting the counter value and the *sid* of the YubiKey with its key k.

$$\textsc{YubiKey} \to \textsc{Server} : pid, \mathsf{senc}(\langle sid, cpt\rangle, k, r)$$

Here, r is the encryption randomness. The server accepts this message if it decrypts with a legitimate key k, and leads to a valid secret value *sid*. Lastly, the counter value obtained through decryption has to be larger than the current value stored in the server database. After this exchange, the server updates its counter with the value just received. We consider the same three security properties as in [19], [26]:

1) *Absence of replay attacks*: the server never accepts for the same YubiKey the same counter twice.
2) *Injective correspondence*: a successful login for the YubiKey *pid* must have been preceded by a button press on this YubiKey for the same counter value, and this counter value is not involved in another successful login.
3) *Monotonicity*: the counter values associated to successful logins are monotonically increasing in time.

As expected, modeling this protocol requires us to rely on our notion of states to store and update counter values on both the YubiKey's and the server's sides.

These three security properties are reachability properties which can be expressed using local meta-formulas, and which actually do not require the use of our new REWRITE-EQUIV inference rule. Furthermore, the security properties 1 and 3 are established relying solely on counter values, while *injective correspondence* requires the use of the INT-CTXT cryptographic assumption. Indeed, it is needed to ensure that the accepted ciphertext has been issued by a legitimate YubiKey.

C. *YubiHSM*

In order to provide some security guarantees even in the case where the server is compromised, Yubico has developed a specific HSM, called YubiHSM. In this setting, the working keys (those used by the YubiKey) are encrypted under a master key mk stored inside the YubiHSM. The YubiHSM lets the server use them to perform some specific cryptographic operations (through a specific API) without ever having access to them in plaintext. Therefore, the interaction between the YubiKey and the server has to go through the HSM (via a secure channel). The HSM receives the otp and the $aead = \mathsf{senc}(\langle k, \langle pid, sid\rangle\rangle, mk, r_0)$, performs the decryption operation, and returns the value of the counter inside the otp to the server. The server, depending on the value of the counter, accepts the exchange or not.

$$\begin{array}{rcll}
\textsc{YubiKey} & \to & \textsc{Server} : & pid, otp \\
\textsc{Server} & \xrightarrow{\mathsf{secure}} & \text{HSM} : & pid, aead, otp \\
\text{HSM} & \xrightarrow{\mathsf{secure}} & \textsc{Server} : & cpt \\
\textsc{Server} & \to & \textsc{YubiKey} : & accept
\end{array}$$

The idea is that since the master key of the YubiHSM is not extractable, the attacker will never learn the value of the working keys (which are protected by mk) even if the server is under the control of the attacker.

We analyze the three same security properties. However, in this new setting, the proof of injective correspondence cannot be carried out in the same fashion, because the INT-CTXT cryptographic assumption on the working keys can no longer be applied. Indeed, these keys occur in plaintext position in, *e.g.*, $aead$. Therefore, to conduct this proof, we proceed in two steps. We first establish the equivalence between the original system and an ideal one, described below. We then prove the security property on the ideal system, using the same proof techniques as in the previous subsection. Thanks to our new rule REWRITE-EQUIV, we put together these two steps and conclude that the security property holds on the original system. In the ideal system, we replace the keys encrypted in $aead$ by dummy keys, so as to be able to rely on the INT-CTXT cryptographic assumption. To maintain the functionality on the ideal system, we replace the dummy keys by the real keys when we need to use them.

To establish the equivalence between these two systems, we first enrich the knowledge of the attacker by revealing additional terms, *e.g.* the secret identifiers and the working keys. Then, we show that the equivalence holds by induction on the length of the trace, with the help of the new BI-DEDUCE rule. Revealing additional information allows us to establish that the additional message outputted during the last step does not provide additional knowledge to the attacker.

VII. CONCLUSION

In order to reason on protocols with mutable states, we have improved both the theoretical foundations and the practical implementation of the SQUIRREL prover. In particular, we have presented a richer and more faithful semantics, an extended proof system that allows a rich interplay between reachability and equivalence properties, and automated deduction techniques based on the notion of bi-deduction.

These improvements have made it possible to carry out new security analyses. We have been able to prove properties of several RFID protocols, overcoming some difficulties tied to a proper modeling of the random oracle model. We have finally carried out the largest SQUIRREL development so far, obtaining the first computational proofs of the YubiKey and YubiHSM protocols.

Future work will obviously include the development of more complex case studies, removing various simplifications in the present ones or tackling more complex protocols. We also plan to improve proof automation, for instance by automating common arguments such as reasonings on counters or stacks of hashes, and by leveraging general-purpose tools such as SMT solvers. We conjecture that our bi-deduction proof system subsumes the FA-DUP rule from [18], which needs to be formally established, and exploited in the implementation of the prover. More generally, we plan to study decidability and completeness issues for our bi-deduction proof system.

REFERENCES

[1] A. Armando, R. Carbone, L. Compagna, J. Cuéllar, G. Pellegrino, and A. Sorniotti, "An authentication flaw in browser-based single sign-on protocols: Impact and remediations," *Computers & Security*, vol. 33, pp. 41–58, 2013.

[2] D. Adrian, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, J. A. Halderman, N. Heninger, D. Springall, E. Thomé, L. Valenta, B. VanderSloot, E. Wustrow, S. Zanella-Béguelin, and P. Zimmermann, "Imperfect forward secrecy: How Diffie-Hellman fails in practice," in *22nd ACM Conference on Computer and Communications Security*, Oct. 2015.

[3] K. Bhargavan, B. Blanchet, and N. Kobeissi, "Verified models and reference implementations for the TLS 1.3 standard candidate," in *2017 IEEE Symposium on Security and Privacy*. IEEE, 2017, pp. 483–502.

[4] S. Meier, B. Schmidt, C. Cremers, and D. A. Basin, "The TAMARIN prover for the symbolic analysis of security protocols," in *CAV*, ser. Lecture Notes in Computer Science, vol. 8044. Springer, 2013, pp. 696–701.

[5] B. Blanchet, "Modeling and Verifying Security Protocols with the Applied Pi Calculus and ProVerif," *Foundations and Trends in Privacy and Security* , vol. 1, no. 1-2, pp. 1 – 135, 10 2016. [Online]. Available: https://hal.inria.fr/hal-01423760

[6] R. Chadha, V. Cheval, Ş. Ciobâcă, and S. Kremer, "Automated verification of equivalence properties of cryptographic protocols," *TOCL*, vol. 17, no. 4, pp. 1–32, 2016.

[7] V. Cheval, S. Kremer, and I. Rakotonirina, "The DEEPSEC prover," in *CAV (2)*, ser. Lecture Notes in Computer Science, vol. 10982. Springer, 2018, pp. 28–36.

[8] K. Bhargavan, A. Bichhawat, Q. Do, P. Hosseyni, R. Küsters, G. Schmitz, and T. Würtele, "DY*: a modular symbolic verification framework for executable cryptographic protocol code," in *EuroS&P 2021-6th IEEE European Symposium on Security and Privacy*, 2021.

[9] B. Blanchet, "A computationally sound mechanized prover for security protocols," in *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2006, pp. 140–154.

[10] G. Barthe, B. Grégoire, S. Heraud, and S. Z. Béguelin, "Computer-aided security proofs for the working cryptographer," in *CRYPTO*, ser. Lecture Notes in Computer Science, vol. 6841. Springer, 2011, pp. 71–90.

[11] D. A. Basin, A. Lochbihler, and S. R. Sefidgar, "CryptHOL: Game-based proofs in higher-order logic," *J. Cryptology*, vol. 33, no. 2, pp. 494–566, 2020.

[12] V. Cortier, S. Kremer, and B. Warinschi, "A survey of symbolic methods in computational analysis of cryptographic systems," *J. Autom. Reasoning*, vol. 46, no. 3-4, pp. 225–259, 2011.

[13] B. Blanchet, "An efficient cryptographic protocol verifier based on prolog rules," in *CSFW*. IEEE Computer Society, 2001, pp. 82–96.

[14] A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuéllar, P. H. Drielsma, P.-C. Héam, O. Kouchnarenko, J. Mantovani *et al.*, "The AVISPA tool for the automated validation of internet security protocols and applications," in *CAV*. Springer, 2005, pp. 281–285.

[15] V. Cheval, V. Cortier, and M. Turuani, "A little more conversation, a little less action, a lot more satisfaction: Global states in proverif," in *31st IEEE Computer Security Foundations Symposium, CSF 2018, Oxford, United Kingdom, July 9-12, 2018*. IEEE Computer Society, 2018, pp. 344–358. [Online]. Available: https://doi.org/10.1109/CSF.2018.00032

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

[17] G. Bana and H. Comon-Lundh, "A computationally complete symbolic attacker for equivalence properties," in *CCS*. ACM, 2014, pp. 609–620.

[18] D. Baelde, S. Delaune, C. Jacomme, A. Koutsos, and S. Moreau, "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, May 2021, p. t.b.d.

[19] R. Künnemann and G. Steel, "Yubisecure? formal security analysis results for the yubikey and yubiHSM," in *Proc. 8th International Workshop on Security and Trust Management (STM'12)*, ser. Lecture Notes in Computer Science, vol. 7783. Springer, 2012, pp. 257–272.

[20] D. Baelde, S. Delaune, A. Koutsos, and S. Moreau, "Cracking the stateful nut – computational proofs of stateful security protocols using the SQUIRREL proof assistant," HAL report 03500056, Tech. Rep., 2022.

[21] G. Bana and H. Comon-Lundh, "Towards unconditional soundness: Computationally complete symbolic attacker," in *POST*, ser. Lecture Notes in Computer Science, vol. 7215. Springer, 2012, pp. 189–208.

[22] M. Ohkubo, K. Suzuki, S. Kinoshita *et al.*, "Cryptographic approach to "privacy-friendly" tags," in *RFID privacy workshop*, vol. 82. Cambridge, USA, 2003.

[23] O. Goldreich, S. Goldwasser, and S. Micali, "How to construct random functions," *J. ACM*, vol. 33, no. 4, pp. 792–807, 1986.

[24] T. van Deursen and S. Radomirovic, "Attacks on RFID protocols," *IACR Cryptol. ePrint Arch.*, vol. 2008, p. 310, 2008. [Online]. Available: http://eprint.iacr.org/2008/310

[25] P. Cousot and R. Cousot, "Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints," in *POPL*. ACM, 1977, pp. 238–252.

[26] R. Künnemann, "Foundations for analyzing security APIs in the symbolic and computational model. (fondations d'analyse de APIs de sécurité dans le modèle symbolique et calculatoire)," Ph.D. dissertation, École normale supérieure de Cachan, France, 2014. [Online]. Available: https://tel.archives-ouvertes.fr/tel-00942459

Using the notations of Figure 2:

$$\begin{aligned}
\mathsf{unfold}_{\mathcal{P}}(m, a[\vec{i}_0]) &= (m_{a[\vec{i}]})\{\vec{i} \mapsto \vec{i}_0\}\\
\mathsf{unfold}_{\mathcal{P}}(m, \mathsf{init}) &= m_{\mathsf{init}}\\
\mathsf{unfold}_{\mathcal{P}}(s[\vec{j}_0], a[\vec{i}_0]) &= (u_{a[\vec{i}],s[\vec{j}]})\{\vec{i} \mapsto \vec{i}_0, \vec{j} \mapsto \vec{j}_0\}\\
\mathsf{unfold}_{\mathcal{P}}(s[\vec{j}_0], \mathsf{init}) &= (u_{0,s[\vec{j}]})\{\vec{j} \mapsto \vec{j}_0\}
\end{aligned}$$

Fig. 7. Interpretation of the partial macro evaluation function $\mathsf{unfold}_{\mathcal{P}}(\cdot,\cdot)$.

A more detailed presentation of the rules of our reachability and general sequents can be found in the long version [20].

APPENDIX A
BI-DEDUCTION

We extend bi-deduction to bi-formulas in a straightforward manner. For any sequence Σ of variables of any sort, protocols $\mathcal{P}_1, \mathcal{P}_2$, set Θ of global meta-formulas, sequences of bi-terms $\#(\vec{u_1}, \vec{u_2})$, bi-formulas $\#(\lambda_1, \lambda_2)$ and $\#(\phi_1, \phi_2)$, we have a bi-deduction judgement:

$$\Sigma;\ \Theta;\ \#(\vec{u_1}, \vec{u_2}) \triangleright_{\mathcal{P}_1,\mathcal{P}_2} \{\#(\lambda_1, \lambda_2) \mid \#(\phi_1, \phi_2)\}.$$

The validity of this judgement is defined in the same way as the validity of a bi-deduction judgement for a bi-term (Equ. (2)): the machine $\mathcal{B}_c$ must compute ϕ_i and $\mathcal{B}_o$ must compute (if ϕ_i then λ_i), i.e. $(\phi_i \Rightarrow \lambda_i)$.

a) Unfold: Before giving our rules, we define the partial macro evaluation function $\mathsf{unfold}_{\mathcal{P}}(\cdot,\cdot)$ in Figure 7.

Left rules

WEAK-L

$$\frac{\Sigma;\ \Theta;\ \vec{u}^{1}_{\#} \rhd \{v_{\#} \mid \phi_{\#}\}}{\Sigma;\ \Theta;\ \vec{u}^{1}_{\#}, \vec{u}^{2}_{\#} \rhd \{v_{\#} \mid \phi_{\#}\}}$$

SEQ-DED

$$\frac{\begin{array}{c}\Sigma;\ \Theta \vdash \tilde{\exists}\vec{\alpha}.\ \left(\begin{array}{l}[\phi_1 \Rightarrow (\psi_1 \wedge u_1 = v_1)]_{\mathcal{P}_1} \tilde{\wedge} \\ {[\phi_2 \Rightarrow (\psi_2 \wedge u_2 = v_2)]_{\mathcal{P}_2}}\end{array}\right) \\ \Sigma;\ \Theta;\ \vec{u}_{\#}, \mathsf{seq}(\vec{\alpha} : \#(u_1, u_2) \mid \#(\psi_1, \psi_2)) \rhd \#(\phi_1, \phi_2)\end{array}}{\Sigma;\ \Theta;\ \vec{u}_{\#}, \mathsf{seq}(\vec{\alpha} : \#(u_1, u_2) \mid \#(\psi_1, \psi_2)) \rhd \{\#(v_1, v_2) \mid \#(\phi_1, \phi_2)\}}$$

PAIR-PROJ

$$\frac{\Sigma;\ \Theta;\ \vec{u}_{\#}, \mathsf{seq}(\vec{\alpha} : u^{1}_{\#} \mid \psi_{\#}), \mathsf{seq}(\vec{\alpha} : u^{2}_{\#} \mid \psi_{\#}) \rhd \{v_{\#} \mid \phi_{\#}\}}{\Sigma;\ \Theta;\ \vec{u}_{\#}, \mathsf{seq}(\vec{\alpha} : \langle u^{1}_{\#}, u^{2}_{\#}\rangle \mid \psi_{\#}) \rhd \{v_{\#} \mid \phi_{\#}\}}$$

UNFOLD-L-1

$$\frac{\begin{array}{c}\Sigma;\ \Theta;\ \vec{u}_{\#}, \#(\mathsf{seq}(\vec{\alpha} : \mathsf{unfold}_{\mathcal{P}_1}(m, \mathsf{a}[\vec{i}]) \mid \psi), u_2) \rhd \{v_{\#} \mid \phi_{\#}\} \\ \Sigma, \vec{\alpha};\ \Theta \vdash [\psi \Rightarrow \mathsf{happens}(\mathsf{a}[\vec{i}])]_{\mathcal{P}_1}\end{array}}{\Sigma;\ \Theta;\ \vec{u}_{\#}, \#(\mathsf{seq}(\vec{\alpha} : m@(\mathsf{a}[\vec{i}]) \mid \psi), u_2) \rhd \{v_{\#} \mid \phi_{\#}\}}$$

UNFOLD-L-2

$$\frac{\begin{array}{c}\Sigma;\ \Theta;\ \vec{u}_{\#}, \#(u_1, \mathsf{seq}(\vec{\alpha} : \mathsf{unfold}_{\mathcal{P}_2}(m, \mathsf{a}[\vec{i}]) \mid \psi)) \rhd \{v_{\#} \mid \phi_{\#}\} \\ \Sigma, \vec{\alpha};\ \Theta \vdash [\psi \Rightarrow \mathsf{happens}(\mathsf{a}[\vec{i}])]_{\mathcal{P}_2}\end{array}}{\Sigma;\ \Theta;\ \vec{u}_{\#}, \#(u_1, \mathsf{seq}(\vec{\alpha} : m@(\mathsf{a}[\vec{i}]) \mid \psi)) \rhd \{v_{\#} \mid \phi_{\#}\}}$$

FRAME

$$\frac{\Sigma;\ \Theta;\ \vec{u}_{\#}, \mathsf{seq}(\vec{\alpha}, T' : \mathsf{frame}@T' \mid \psi_{\#} \wedge T' \leq T) \rhd \{v_{\#} \mid \phi_{\#}\}}{\Sigma;\ \Theta;\ \vec{u}_{\#}, \mathsf{seq}(\vec{\alpha} : \mathsf{frame}@T \mid \psi_{\#}) \rhd \{v_{\#} \mid \phi_{\#}\}}$$

Right rules

SPLIT-COND

$$\frac{\Sigma;\ \Theta;\ \vec{u}_{\#} \rhd \{v_{\#} \mid \psi_{\#}\} \qquad \Sigma;\ \Theta;\ \vec{u}_{\#} \rhd \{\phi_{\#} \mid \psi_{\#}\}}{\Sigma;\ \Theta;\ \vec{u}_{\#} \rhd \{v_{\#} \mid \phi_{\#} \wedge \psi_{\#}\}}$$

FA

$$\frac{\Sigma;\ \Theta;\ \vec{u}_{\#} \rhd \{\vec{v}_{\#} \mid \phi_{\#}\}}{\Sigma;\ \Theta;\ \vec{u}_{\#} \rhd \{f(\vec{v}_{\#}) \mid \phi_{\#}\}}$$

FA$_0$

$$\frac{\Sigma;\ \Theta;\ \vec{u}_{\#} \rhd \phi_{\#}}{\Sigma;\ \Theta;\ \vec{u}_{\#} \rhd \{f() \mid \phi_{\#}\}}$$

FA-ITE

$$\frac{\begin{array}{c}\Sigma;\ \Theta;\ \vec{u}_{\#} \rhd \{\psi_{\#} \mid \phi_{\#}\} \\ \Sigma;\ \Theta;\ \vec{u}_{\#} \rhd \{t_{\#} \mid \phi_{\#} \wedge \psi_{\#}\} \\ \Sigma;\ \Theta;\ \vec{u}_{\#} \rhd \{t'_{\#} \mid \phi_{\#} \wedge \neg\psi_{\#}\}\end{array}}{\Sigma;\ \Theta;\ \vec{u}_{\#} \rhd \{\mathsf{if}\ \psi_{\#}\ \mathsf{then}\ t_{\#}\ \mathsf{else}\ t'_{\#} \mid \phi_{\#}\}}$$

FA-FIND

$$\frac{\begin{array}{c}\Sigma, \vec{i};\ \Theta;\ \vec{u}_{\#} \rhd \{\psi_{\#} \mid \phi_{\#}\} \\ \Sigma, \vec{i};\ \Theta;\ \vec{u}_{\#} \rhd \{t_{\#} \mid \phi_{\#} \wedge \psi_{\#}\} \\ \Sigma;\ \Theta;\ \vec{u}_{\#} \rhd \{t'_{\#} \mid \phi_{\#} \wedge \forall\vec{i}, \neg\psi_{\#}\}\end{array}}{\Sigma;\ \Theta;\ \vec{u}_{\#} \rhd \{\mathsf{find}\ \vec{i}\ \mathsf{suchthat}\ \psi_{\#}\ \mathsf{in}\ t_{\#}\ \mathsf{else}\ t'_{\#} \mid \phi_{\#}\}}$$

FA-$\mathcal{Q}$

$$\frac{\Sigma, \alpha;\ \Theta;\ \vec{u}_{\#} \rhd \{v_{\#} \mid \phi_{\#}\}}{\Sigma;\ \Theta;\ \vec{u}_{\#} \rhd \{\mathcal{Q}\alpha.v_{\#} \mid \phi_{\#}\}}\ \mathcal{Q} \in \{\exists, \forall\}$$

FA-$\vee$-L

$$\frac{\begin{array}{c}\Sigma;\ \Theta;\ \vec{u}_{\#} \rhd \{\psi^{0}_{\#} \mid \phi_{\#}\} \\ \Sigma;\ \Theta;\ \vec{u}_{\#} \rhd \{\psi^{1}_{\#} \mid \phi_{\#} \wedge \neg\psi^{0}_{\#}\}\end{array}}{\Sigma;\ \Theta;\ \vec{u}_{\#} \rhd \{\psi^{0}_{\#} \vee \psi^{1}_{\#} \mid \phi_{\#}\}}$$

FA-$\vee$-R

$$\frac{\begin{array}{c}\Sigma;\ \Theta;\ \vec{u}_{\#} \rhd \{\psi^{1}_{\#} \mid \phi_{\#}\} \\ \Sigma;\ \Theta;\ \vec{u}_{\#} \rhd \{\psi^{0}_{\#} \mid \phi_{\#} \wedge \neg\psi^{1}_{\#}\}\end{array}}{\Sigma;\ \Theta;\ \vec{u}_{\#} \rhd \{\psi^{0}_{\#} \vee \psi^{1}_{\#} \mid \phi_{\#}\}}$$

FA-$\wedge$-L

$$\frac{\Sigma;\ \Theta;\ \vec{u}_{\#} \rhd \{\psi^{1}_{\#} \mid \phi_{\#} \wedge \psi^{0}_{\#}\}}{\Sigma;\ \Theta;\ \vec{u}_{\#} \rhd \{\psi^{0}_{\#} \wedge \psi^{1}_{\#} \mid \phi_{\#}\}}$$

FA-$\wedge$-R

$$\frac{\Sigma;\ \Theta;\ \vec{u}_{\#} \rhd \{\psi^{0}_{\#} \mid \phi_{\#} \wedge \psi^{1}_{\#}\}}{\Sigma;\ \Theta;\ \vec{u}_{\#} \rhd \{\psi^{0}_{\#} \wedge \psi^{1}_{\#} \mid \phi_{\#}\}}$$

FA-$\Rightarrow$

$$\frac{\begin{array}{c}\Sigma;\ \Theta;\ \vec{u}_{\#} \rhd \{\psi^{0}_{\#} \mid \phi_{\#}\} \\ \Sigma;\ \Theta;\ \vec{u}_{\#} \rhd \{\psi^{1}_{\#} \mid \phi_{\#} \wedge \psi^{0}_{\#}\}\end{array}}{\Sigma;\ \Theta;\ \vec{u}_{\#} \rhd \{\psi^{0}_{\#} \Rightarrow \psi^{1}_{\#} \mid \phi_{\#}\}}$$

FA-$\neg$

$$\frac{\Sigma;\ \Theta;\ \vec{u}_{\#} \rhd \{\psi_{\#} \mid \phi_{\#}\}}{\Sigma;\ \Theta;\ \vec{u}_{\#} \rhd \{\neg\psi_{\#} \mid \phi_{\#}\}}$$

FA-$\dagger$

$$\frac{\Sigma;\ \Theta;\ \vec{u}_{\#} \rhd \phi_{\#}}{\Sigma;\ \Theta;\ \vec{u}_{\#} \rhd \{\dagger \mid \phi_{\#}\}}\ \dagger \in \{\top, \bot\}$$

FA-SEQ

$$\frac{\Sigma, \vec{\alpha};\ \Theta;\ \vec{u}_{\#} \rhd \{v_{\#} \mid \phi_{\#} \wedge \psi_{\#}\}}{\Sigma;\ \Theta;\ \vec{u}_{\#} \rhd \{\mathsf{seq}(\vec{\alpha} : v_{\#} \mid \psi_{\#}) \mid \phi_{\#}\}}$$

PURE-DED

$$\frac{v, \phi \text{ pure trace formula}}{\Sigma;\ \Theta;\ \vec{u}_{\#} \rhd \{\#(v, v) \mid \#(\phi, \phi)\}}$$

UNFOLD-R-1

$$\frac{\begin{array}{c}\Sigma;\ \Theta;\ \vec{u}_{\#} \rhd \{\#(\mathsf{unfold}_{\mathcal{P}_1}(m, \mathsf{a}[\vec{i}]), v_2) \mid \phi_{\#}\} \\ \Sigma;\ \Theta \vdash [\phi_1 \Rightarrow \mathsf{happens}(\mathsf{a}[\vec{i}])]_{\mathcal{P}_1}\end{array}}{\Sigma;\ \Theta;\ \vec{u}_{\#} \rhd \{\#(m@(\mathsf{a}[\vec{i}]), v_2) \mid \#(\phi_1, \phi_2)\}}$$

UNFOLD-R-2

$$\frac{\begin{array}{c}\Sigma;\ \Theta;\ \vec{u}_{\#} \rhd \{\#(v_1, \mathsf{unfold}_{\mathcal{P}_2}(m, \mathsf{a}[\vec{i}])) \mid \phi_{\#}\} \\ \Sigma;\ \Theta \vdash [\phi_2 \Rightarrow \mathsf{happens}(\mathsf{a}[\vec{i}])]_{\mathcal{P}_2}\end{array}}{\Sigma;\ \Theta;\ \vec{u}_{\#} \rhd \{\#(v_1, m@(\mathsf{a}[\vec{i}])) \mid \#(\phi_1, \phi_2)\}}$$

UNFOLD-R-SYM

$$\frac{\Sigma;\ \Theta;\ \vec{u}_{\#} \rhd \{\#(\mathsf{unfold}_{\mathcal{P}_1}(m, \mathsf{a}[\vec{i}]), \mathsf{unfold}_{\mathcal{P}_2}(m, \mathsf{a}[\vec{i}])) \mid \phi_{\#}\}}{\Sigma;\ \Theta;\ \vec{u}_{\#} \rhd \{m@(\mathsf{a}[\vec{i}]) \mid \phi_{\#}\}}$$

Symmetric rule

CASE

$$\frac{\left(\Sigma, \vec{i};\ \Theta\{\tau \mapsto \mathsf{a}[\vec{i}]\};\ \vec{u}_{\#}\{\tau \mapsto \mathsf{a}[\vec{i}]\} \rhd \{v_{\#}\{\tau \mapsto \mathsf{a}[\vec{i}]\} \mid \phi_{\#}\{\tau \mapsto \mathsf{a}[\vec{i}]\}\}\right)_{\mathsf{a} \in \mathcal{A}}}{\Sigma, \tau;\ \Theta;\ \vec{u}_{\#} \rhd \{v_{\#} \mid \phi_{\#}\}}$$

Conventions: $\mathcal{A}$ is the set of action names of $\mathcal{P}_1, \mathcal{P}_2$. When necessary, we use the fact that $u = \mathsf{seq}(\epsilon\colon u \mid \top)$ to apply a rule. We assume that the environment Σ is only extended by fresh variables (*e.g.* in FA-$\mathcal{Q}$, we assume that $\alpha \notin \Sigma$). This is always possible through alpha-renaming. The FA rule only applies when f is of non-zero arity.

Fig. 8. Bi-deduction rules for protocols $\mathcal{P}_1, \mathcal{P}_2$.

b) Rules: We present the bi-deduction rules of Figure 8, which are grouped in three categories, depending on whether they act on the left, right, or both sides of the $\triangleright$ symbol.

We start by describing left rules. The WEAK-L rule states that if $v_\#$ can be bi-deduced from $\vec{u}^1_\#$, then it can bi-deduced from $\vec{u}^1_\#, \vec{u}^2_\#$. SEQ-DED allows to bi-deduce a bi-term $\#(v_1, v_2)$, whenever $\#(\phi_1, \phi_2)$ holds, from $\vec{u}_\#, \mathsf{seq}(\vec{\alpha} : \#(u_1, u_2) \mid \#(\psi_1, \psi_2))$ if: i) $\#(\phi_1, \phi_2)$ can be bi-deduced using the allowed bi-terms; and ii), the bi-term $\#(v_1, v_2)$ appears in the sequence *simultaneously* on both sides: there must exists $\vec{\alpha}$ such that, for any $i \in \{1, 2\}$, if ϕ_i holds then $u_i = v_i$ and the sequence condition ψ_i holds. PAIR-PROJ allows to exploit the fact that a machine can recover a pair's components. Using the UNFOLD-L-I rules, we can replace a macro by its definition, as long as it is at a timestamp that happens. The FRAME rule states that $\mathsf{frame}@T$ contains all $\mathsf{frame}@T'$ for any $T' \leq T$.

We now describe right rules. The rule SPLIT-COND allows to weaken the condition on the right-hand side when it is bi-deducible. The rule FA says that the image of a function $f(\vec{v}_\#)$ is bi-deducible whenever its arguments $\vec{v}_\#$ are bi-deducible. In case f is of arity zero, we must also check that the condition $\phi_\#$ is bi-deducible — see rule FA$_0$. We then have many variants of the FA rule, for other logical constructs that are translated, in the base logic, to function symbol applications: FA-ITE for conditionals; FA-FIND for lookups; FA-$\mathcal{Q}$ for quantification over indices or timestamps; FA-$\vee$, FA-$\wedge$, FA-$\Rightarrow$, FA-$\neg$ and FA-$\dagger$ for boolean connectives; and FA-SEQ for sequences. Remark that, whenever possible, we restrict the set of terms that must be deduced by adding new conditions. *E.g.*, in the FA-ITE rule, we know that we must deduce the then branch only if the condition ψ holds. PURE-DED states that pure trace formulas, *i.e.* formulas that only talk about the trace model, are always bi-deducible. Indeed, once the trace model is fixed, they have a constant value (either always true or always false). The UNFOLD-R-I rules allow to replace, on the right-hand side, a macro $m@(\mathsf{a}[\vec{i}])$ by its definition, as long as it is at a timestamp that happens. We have another unfolding rule on the right-hand side. The UNFOLD-R-SYM rules allow to replace a macro $m@(\mathsf{a}[\vec{i}])$ by its definition of *both* sides of the bi-term. Remark that we do not need to check that $\mathsf{a}[\vec{i}]$ happens: indeed, if it does, then by definition $m@(\mathsf{a}[\vec{i}]) = \mathsf{empty}$ (simultaneously on both sides of the bi-term), which is a constant, and is therefore bi-deducible.

Finally, the symmetric CASE rule allows to do a case analysis.

Appendix B
Automatic Inference of Bi-Deduction Invariants

Assume we want to prove the bi-deduction judgement $\Sigma;\ \Theta;\ u_\# \triangleright v_\#$. We give a high-level description of how we infer the invariant $U_\#$ of Section V-E.

Consider a fixed set of invariant candidates $\mathcal{I}$ of the form $U^i_\# \stackrel{\text{def}}{=} \lambda\tau.u^i_\#[\tau]$, where $U^i_\#$ free variables are bound by Σ. Our high-level description is agnostic in the precise set of invariant candidates $\mathcal{I}$, as long as it supports some operations (discussed below). Still, to help the reader, we give an example of such a set which can be used to instantiate our algorithm.

Example 18. *A state slice is an element of the form:*

$$\lambda\tau.\,\mathsf{seq}(\vec{j} : s[\vec{i}]@\tau)$$

where $\vec{i} \subseteq \vec{j} \cup \Sigma$*. Then, a state invariant candidate is any finite union of state slices, and we let* $\mathcal{I}_S$ *be the set of state invariant candidates.*

Our inductive invariant procedure computes a sequence of elements $(U^i_\#)_{i \leq N} \in \mathcal{I}^{N+1}$ such that:

(A) for all i, $U^{i+1}_\#[\tau]$ can be deduced from the initial knowledge and the known terms at previous steps:
$$\Sigma, \tau;\ \Theta;\ u_\#, \mathsf{seq}(\tau' \colon U^i_\#[\tau'] \mid \tau' < \tau) \triangleright U^{i+1}_\#[\tau]$$

(B) the candidate invariants are monotonously decreasing:
$$\Sigma, \tau;\ \Theta;\ U^i_\#[\tau] \triangleright U^{i+1}_\#[\tau] \qquad \text{(for all } i\text{)}$$

Our procedure stops as soon as it computes a candidate invariant $U^N_\#$ which is smaller than $U^{N-1}_\#$ w.r.t. $\triangleright$. More precisely:

(C) the procedure stops at step $N > 0$ if it reached a post fix-point: $\Sigma, \tau;\ \Theta;\ u_\#, U^N_\#[\tau] \triangleright U^{N-1}_\#[\tau]$

We can prove that if these conditions hold, then $U^N_\#$ is an inductive invariant.

Proposition 4. *If a sequence* $(U^i_\#)_{i \leq N} \in \mathcal{I}^{N+1}$ *satisfies conditions (A) to (C), then* $U^N_\#$ *is an invariant of the protocol:*

$$\Sigma, \tau;\ \Theta;\ u_\#, \mathsf{seq}(\tau' \colon U^N_\#[\tau'] \mid \tau' < \tau) \triangleright U^N_\#[\tau] \qquad (\star)$$

We now put everything together.

Lemma 1. *If a sequence* $(U^i_\#)_{i \leq N} \in \mathcal{I}^{N+1}$ *satisfies conditions (A) to (C), and if:*

$$\Sigma;\ \Theta;\ u_\#, \mathsf{seq}(\tau \colon U_\#[\tau]) \triangleright v_\#$$

then $\Sigma;\ \Theta;\ u_\# \triangleright v_\#$.

Instantiating $\mathcal{I}$. To instantiate this procedure with some set of candidate invariants $\mathcal{I}$, we require the following.

- There exists an effective procedure that can compute, for any initial knowledge $u_\#$ and for any $U_\#$ in $\mathcal{I}$, a candidate invariant $V_\# \in \mathcal{I}$ such that:
$$\Sigma, \tau;\ \Theta;\ u_\#, \mathsf{seq}(\tau' \colon U_\#[\tau'] \mid \tau' < \tau) \triangleright V_\#[\tau]$$
and $\Sigma, \tau;\ \Theta;\ U_\#[\tau] \triangleright V_\#[\tau]$. This procedure allows to compute the decreasing sequence of candidate invariants $(U^i_\#)_{i \leq N}$ satisfying conditions (A) and (B).
- There exists a sound (though not necessarily complete) procedure that can check, for any $U_\#, V_\#$ in $\mathcal{I}$, if: $\Sigma, \tau;\ \Theta;\ U_\#[\tau] \triangleright V_\#[\tau]$. This is used to verify the termination condition (C).
- To ensure termination, any sequence $(U^i_\#)_i \in \mathcal{I}^\infty$ such that conditions (A) and (B) holds must be stationnary.[1]

We have designed and implemented such procedures for the set $\mathcal{I}_S$ of state invariants introduced in Example 18. We do not detail them here.

[1] Sets of invariant candidates $\mathcal{I}$ which do not have this property could still be used, by adapting the procedure we presented so far with a widening operator [25].