A Computationally Complete Symbolic Attacker for Equivalence Properties

Gergei Bana*
INRIA Paris-Rocquencourt
Paris, France
bana@math.upenn.edu

Hubert Comon-Lundh†
LSV, ENS Cachan
Cachan, France
comon@lsv.ens-cachan.fr

ABSTRACT

We consider the problem of computational indistinguishability of protocols. We design a symbolic model, amenable to automated deduction, such that a successful inconsistency proof implies computational indistinguishability. Conversely, symbolic models of distinguishability provide clues for likely computational attacks. We follow the idea we introduced earlier for reachability properties, axiomatizing what an attacker cannot violate. This results a computationally complete symbolic attacker, and ensures unconditional computational soundness for the symbolic analysis. We present a small library of computationally sound, modular axioms, and test our technique on an example protocol. Despite additional difficulties stemming from the equivalence properties, the models and the soundness proofs turn out to be simpler than they were for reachability properties.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols—*Protocol verification*

Keywords

Security Protocols; Protocol Indistinguishability; Symbolic Verification; Computational Soundness

1. INTRODUCTION

Symbolic analysis of security protocols is a well-established research direction. Attacker capabilities in such a framework are modeled by the so-called "Dolev-Yao attacker", which can only perform sequences of actions from a fixed set of possible actions. There are several automated tools based on this framework, including PROVERIF [13], SCYTHER [22], AVISPA [24]. When such a tool claims that there is no attack, we must be careful: this only means there is no attack in the model that is considered. There could be attacks relying on some actions that are not accounted for by the Dolev-Yao attacker, for instance, exploiting imperfections of cryptographic primitives (e.g.,[25]).

Closer to reality than the Dolev-Yao model is the so-called computational model, which uses complexity theory to express security properties. Attackers are modeled as probabilistic polynomial-time algorithms. It is far more convincing when a protocol is proved secure in this model than with respect to the Dolev-Yao model.

In order to overcome this discrepancy, the following directions have been considered in the literature:

- prove *computational soundness results*, stating that the Dolev-Yao model is fully abstract with respect to the computational one, hence any attack that can be carried out on the computational implementation is also possible in the Dolev-Yao model. There is a long line of research in this direction, starting with [2, 4].
- carry out the proof in a computational model. Since this is very long and error-prone, it should be, at least partly, carried out with the help of a formal prover. CRYPTOVERIF [14] and EASYCRYPT [10] are examples of provers, which complete the proofs in the computational setting.

Both directions have several drawbacks. Computational soundness results require very strong assumptions, often unrealistic, and their proofs are themselves very long and technical. Their scope is limited; for instance, in the case of equivalence properties (which is the subject our paper), there are very few soundness results for active attackers, notably [18, 21], and their assumptions are disputable. In particular, all (unless KDM secure encryption is assumed) soundness proofs require the absence of key cycles and that the keys are honestly generated (except [19]). A further problem of such soundness results is that they are not modular: including new primitives requires a soundness proof for the whole system again.

Proving the protocols directly in the computational setting is (currently) often not possible automatically due to the complexity of the task. Moreover, we do not know what to do when the current provers fail to prove a protocol: is there a flaw in the protocol or are we and the prover simply not capable enough to complete the proof?

Our previous work, presented in [6], takes a different approach. We also formalize the protocols and the attackers in a symbolic way. However, instead of specifying explicitly everything that is possible (as the Dolev-Yao approach does) for the attacker and hence considering the least model, we specified what cannot be violated and hence considered the greatest model that the set of rules permit. The properties that cannot be violated (because of the na-

*This work has been partially supported by the ANR project *ProSe* (decision ANR-2010-VERS-004) and the ERC project *CRYSP* (259639) as well as the FCT project *ComFormCrypt* (PTDC/EIA-CCO/113033/2009).

†This work has been partially supported by the ANR project *ProSe* (decision ANR-2010-VERS-004)

CCS'14, November 3–7, 2014, Scottsdale, Arizona, USA.

http://dx.doi.org/10.1145/2660267.2660276.

ture of PPT algorithms, because of assumptions on the primitives, etc.) are called axioms. The attacker may perform any action that is consistent with the specification. If one manages to prove the protocol in such a setting, then it is guaranteed that the protocol is computationally secure, roughly because all models are considered that do not violate the axioms, which includes the computational model. In other words, to every successful computational attack, there is a corresponding successful symbolic attack: all computational attacks are symbolically accounted for, and that is why we call our symbolic attacker *computationally complete symbolic attacker.* Therefore, we reduced protocol security to an inconsistency proof for a set of first-order formulas: *if the negation of the security formula is inconsistent with the set of axioms, then the protocol is secure in any model of the axioms.*

On the other hand, a model for the negation of the security formula together with the axioms is a clue: from such a model we can derive missing properties that are necessary for the protocol security. This is an advantage over the direct methods.

Yet another advantage: with the our technique, we do not have to assume (for instance) that the protocol does not produce key cycles or that the keys are honestly generated. Such properties may be necessary for the conclusions of some axioms, but this does not change the scope of the method: if we complete an inconsistency proof, then the protocol is secure, even if it produces key cycles or uses dishonest keys etc... Basically, what is shown on the way is that such occurrences of key-cycles, dishonest keys etc. are harmless for the security.

This very seducing approach has itself some difficulties and limitations, *e.g.*:

- Axioms have to be designed. They are independent of the protocol. Such formulas formally state the cryptographic assumptions, or, more precisely, properties implied by the cryptographic assumptions. They can be re-used for several protocol analysis. There were several axioms (independent of protocols) proposed by Bana *et al.* in [5, 7], actual protocols were verified with them, and new attacks were found. However, handling further primitives/new security assumptions requires new axioms.

- Consistency proofs have to be automated. Scerri *et al.* are in the process of developing a tool. In principle, it is possible to complete an automated consistency check in an efficient way [20].

- We only consider protocols with a finite number of control states, ruling out replicated processes.[1]

- In our previous work, we investigated reachability properties only. This was a weak point not only because for instance privacy cannot be expressed as a reachability property, but also because cryptographic assumptions are nowadays expressed using indistinguishability, making the search for convenient axioms difficult.

[1]We could easily drop this assumption (as in [5, 7]), however allowing on the computational side only attacks each of which has a bound on the number of exchanged messages (independently of the security parameter as an initial input). The best we can expect from a symbolic model that does not refer explicitly to the security parameter is to capture computational attacks with such a bound. In particular, none of the existing soundness results in the literature capture attacks that do not have this bound. We choose here to include this assumption as we do not loose much, but the formulation becomes much simpler.

The present paper intends to fill the last weakness. We show how it is possible to apply the "greatest model" approach to equivalence properties. In some aspects the case of equivalence properties has turned out to be simpler than that for reachability properties. The (unconditional) computational soundness result is quite straightforward. The axioms are more natural, they correspond more closely to the computational assumptions on the cryptographic primitives.

Before explaining the details, let us first show on simple examples why it is not straightforward to extend our previous work on reachability to equivalence properties.

1.1 Difficulties of the extension to equivalence properties

Two protocols P_1, P_2 are computationally indistinguishable if for each probabilistic polynomial-time attacker, the probability that it outputs 1 when interacting with P_1 and the probability that it outputs 1 when interacting with P_2 are only negligibly different.

In order to prove such a property without any explicit reference to the security parameter, the first idea that comes to mind is to consider the symbolic executions of the protocols P_1 and P_2, and, for each trace t_1 of P_1, look for a trace t_2 of P_2 such that t_1 and t_2 are computationally indistinguishable, and vice versa. The following example shows that this fails, even in the simplest cases:

EXAMPLE 1. Consider the two processes (b is assumed here to be 0 or 1; we could also consider the first bit of a nonce).

$$\nu b.\ \mathsf{out}(b) \qquad\qquad \nu b.\mathsf{if}\ b = 0\ \mathsf{then}\ \mathsf{out}(1)\ \mathsf{else}\ \mathsf{out}(0)$$

The first process generates a random bit b and outputs it. The second generates first a random bit b, and if b is 0, it outputs 1, if b is 1, it outputs 0. Clearly, both processes output a random bit, so they are computationally indistinguishable as the distributions are identical. However, symbolically there is only one trace for the first process, while two for the second, and both traces of the second process (one where the output is 1 and one where the output is 0) can be distinguished from the single trace of the first one (where the output is 0 or 1 with probability $1/2$). That is, there is a trace t_1 of P_1 such that for any t_2 of P_2, t_1 and t_2 are distinguishable. □

Clearly, the problem in the above example is that in the left protocol a single symbolic trace covers all computational traces, while in the right protocol, a single symbolic trace covers half of the computational traces. So from this (trivial) example, one may think that it suffices to split the conditions in either (or both) processes so that symbolic traces on the two sides cover corresponding parts of the computational traces as, for instance, this is done in the symbolic verification algorithm of [17]. It is not clear however how to perform such a splitting systematically. Consider for instance the following modification of the previous example:

EXAMPLE 2. Consider the two processes:

$$\nu n'.\ \mathsf{out}(n') \qquad \nu b.\nu n.\mathsf{if}\ b = 0\ \mathsf{then}\ \mathsf{out}(1 \cdot n)\ \mathsf{else}\ \mathsf{out}(0 \cdot n)$$

The dot means concatenation. Assume also that n is drawn uniformly at random in $\{0,1\}^\eta$, n' is drawn uniformly at random in $\{0,1\}^{\eta+1}$ and b is drawn uniformly at random in $\{0,1\}$. The processes are indistinguishable, since they output a random bit string of length $\eta + 1$. Matching the branches of the two processes would require to distinguish cases depending on the values of the first bit of n'. □

It is unclear how to perform the above splitting automatically: there is no b like in the first process and therefore we cannot branch, depending on a test on b. Guessing the necessary additional tests requires a deep insight on the distributions of the output.

The next difficulty is a standard one when moving from reachability properties to equivalence properties. Note that even if an attacker interacting with P_1 can get the same information as an attacker interacting with P_2, this does not imply that P_1 and P_2 are indistinguishable, since the information might be obtained in different ways in the two experiments. A single attacker though, using the same recipes in both experiments, may notice a difference in the results. The other direction is also true: attackers using different recipes may obtain different results in the two processes while the processes are in fact equivalent.

The procedures that are designed to check trace equivalence in the usual symbolic model ensure the same attacker's actions for both protocols, either by overlapping the protocols and hence reducing the equivalence to a reachability property of a bi-process [15], or else by explicitly representing the attacker's actions (e.g. [17]). The first solution actually checks a stronger indistinguishability property, which prevents us from proving the equivalence of two processes that have different control structures such as the simple example of the private authentication protocol that we will present in this paper. The second solution specifies all computations the attacker can do, which is precisely what we want to avoid here (following the idea in [6]), because the absence of such an explicit description is necessary to make the symbolic attacker computationally complete.

EXAMPLE 3.

$$P_1 = \mathsf{out}(\langle a, b\rangle).\mathsf{in}(x).\mathsf{if}\ x = a\ \mathsf{then}\ 0\ \mathsf{else}\ 1$$
$$P_2 = \mathsf{out}(\langle b, a\rangle).\mathsf{in}(y).\mathsf{if}\ y = b\ \mathsf{then}\ 0\ \mathsf{else}\ 1$$

P_1 and P_2 are clearly computationally equivalent if the names a, b are indistinguishable, and should be in our symbolic model as well. In both cases, x (resp. y) is any message that can be computed by the attacker from the output. The attacker has however to use the same recipe in both cases, which requires an explicit reference to the recipe: it is not possible to continue the idea of [6] and just say that x (resp. y) is computed by the attacker from the previous outputs since, in that case, we could choose for x the first projection and for y the second projection ($x = y = a$) which yields an output 0 in the first process and an output 1 in the second process. □

This example reminds us that, when checking equivalence properties, the *same* attacker is faced with the two experiments.

1.2 Our contributions

We use three simple tricks to solve the difficulties above:

- Instead of trying to match the execution branches in the two protocols, *we fold the protocols, including the control structure into the message terms* so that each protocol has only one trace. This trick is inspired by the *merging* of the process branches defined in [16]. In that paper, the authors' goal was to use the PROVERIF verification tool on protocols that have different control structures. Equivalences are checked in PROVERIF by overlapping the processes, computing a so-called bi-process, and checking that they do not diverge at any point. This yields (false) attacks, as soon as the protocols have different control structures. To overcome this difficulty, the trick of [16] is that at each step, instead of various possible outputs on the various branches, the output is a single term that includes a conditional branching in the form of function symbols, while the conditional branching was part of the control structure of the original process. We do the same. We adapt the merging procedure of [16] to our aims, and prove that the merging is correct in the sense that it yields the same executions, whatever execution model is considered. For instance, if we come back to Example 2, the second process is folded into

  $$\nu b.\nu n.\mathsf{out}(\ \mathsf{if}\ b = 0\ \mathsf{then}\ 1 \cdot n\ \mathsf{else}\ 0 \cdot n\ )$$

  introducing $\mathsf{if}\ \_\ \mathsf{then}\ \_\ \mathsf{else}\ \_$ as a function symbol on terms.

- Once the branches are merged, the output of each experiment is just a single sequence of messages, whose indistinguishability we express using the predicate $\sim$.[2] In the previous example, we are left to check the indistinguishability of terms:

  $$n' \sim \mathsf{if}\ b = 0\ \mathsf{then}\ 1 \cdot n\ \mathsf{else}\ 0 \cdot n$$

- In order to express that it is the same computation that is performed by the attacker on each of the experiments, we simply use free function symbols. The same function symbols are used in both experiments, when the computations need to be identical. This straightforward idea works in our context because security is proved by checking the inconsistency of a formula. First-order inconsistency means that there is no first-order model, in particular no interpretation (the same for both experiments) of the these free function symbols (no attacker's computations) that violates the security property. For instance in Example 3, the equivalence of the two processes is expressed as

  $$\mathsf{if}\, g(\langle a, b\rangle) = a\, \mathsf{then}\, 0\, \mathsf{else}\, 1 \sim \mathsf{if}\, g(\langle b, a\rangle) = b\, \mathsf{then}\, 0\, \mathsf{else}\, 1$$

  for any free function symbol g.

These three very simple ideas are actually sufficient for reducing the computational indistinguishability to the unsatisfiability of a (recursive) set of first-order formulas. Then, axioms have to be introduced for the $\sim$ predicate, some of which depend on the security assumptions on the cryptographic primitives. In Section 7 we deliver some examples of axioms and prove their soundness. We illustrate our method in Section 8, showing how it is applied to prove the computational security of a version of the private authentication protocol [3]. We also show an attack on the flawed original version of the protocol.

The layout is the following: In Section 2 we define the syntax of our formulas; we only consider equivalences or their Boolean combination. In Section 3 we explain how such formulas are interpreted. This includes the computational semantics, which is now much simpler than it was in case of a computability predicate (as in [6]). In Section 4 we define the protocols: they are arbitrary finite labeled transition systems. In Section 5 we discuss the folding procedure. In Section 6 we state and prove the unconditional soundness result. In Section 7 we provide some axioms that are proved to be computationally sound and that are sufficient for proving the correct version of the private authentication protocol in Section 8.

2. SYNTAX

2.1 Terms

Let S be a finite set of *sorts* that includes at least the sorts bool and $\mathsf{message}$. Function symbols: $\mathcal{F}$ is any (fixed) set of function symbols together with a type in $S^* \times S$. When $\mathsf{type}(f) = (s_1, \ldots, s_n, s)$, we also write $f : s_1 \times \ldots \times s_n \to s$ and call n the *arity* of f. $\mathcal{F}$ includes at least

[2]Note, the formal indistinguishability relation [8, 23] is not a predicate, it is a model with a set of terms as its domain.

- conditional branching $\mathsf{if}\ \_\ \mathsf{then}\ \_\ \mathsf{else}\ \_$: $\mathsf{bool} \times \mathsf{message} \times \mathsf{message} \to \mathsf{message}$,
- Booleans $\mathsf{true} :\to \mathsf{bool}$ and $\mathsf{false} :\to \mathsf{bool}$
- empty message $\mathbf{0} :\to \mathsf{message}$.
- equality test $\mathtt{EQ}(\_) : \mathsf{message} \times \mathsf{message} \to \mathsf{bool}$
- lengths agreement test $\mathtt{EQL}(\_) : \mathsf{message} \times \mathsf{message} \to \mathsf{bool}$

This is pure syntax. For instance true and false may be interpreted as PPT algorithms, as we will see later; the sort bool may be interpreted as a set that has more than 2 elements (as boolean functions).

EXAMPLE 4. We are going to illustrate our method on the private authentication protocol [3]. This protocol does the following: First some initiator agent a' sends a message $\{\mathsf{pk}_{a'}, n_0\}^{r_0}_{\mathsf{pk}_b}$ to responder b, where pk_x denotes the public key of agent x, n_0 is a freshly generated nonce, r_0 is a freshly generated random input, and the curly brackets denote an encryption. Agent b checks if the first projection of the decryption of what it received is pk_a with a fixed agent a, and if yes, then it sends back $\{n_0, n\}^r_{\mathsf{pk}_a}$ with a fresh nonce n and a fresh randomness r. Otherwise it does not send anything (outputs $\mathbf{0}$). We can sketch this as

1. $a' \to b$: $\{\mathsf{pk}_{a'}, n_0\}^{r_0}_{\mathsf{pk}_b}$
2. $b \xrightarrow{\text{if } \mathsf{pk}_a = \mathsf{pk}_{a'}} a'$: $\{n_0, n\}^r_{\mathsf{pk}_a}$

Accordingly, the formalization of this protocol relies on the following function symbols (other than true, false, $\mathbf{0}$, $\mathtt{EQ}(\_)$, $\mathtt{EQL}(\_)$, $\mathsf{if}\ \_\ \mathsf{then}\ \_\ \mathsf{else}\ \_$):

$$
\begin{array}{lrcl}
\{\_\}_\_ : & \mathsf{message} \times \mathsf{message} \times \mathsf{message} & \to & \mathsf{message} \\
\mathsf{dec}(\_,\_) : & \mathsf{message} \times \mathsf{message} & \to & \mathsf{message} \\
\mathsf{k}(\_) : & \mathsf{message} & \to & \mathsf{message} \\
\mathsf{pk}(\_), \mathsf{sk}(\_) : & \mathsf{message} & \to & \mathsf{message} \\
\langle \_,\_ \rangle : & \mathsf{message} \times \mathsf{message} & \to & \mathsf{message} \\
\pi_1(\_), \pi_2(\_) : & \mathsf{message} & \to & \mathsf{message}
\end{array}
$$

Here, $\mathsf{dec}(x, y)$ denotes decryption of x with secret key y. $\mathsf{k}(x)$ is the public-key secret-key pair of agent x, $\mathsf{pk}(x)$ is the public key, $\mathsf{sk}(x)$ is the secret key part of key x. We use the shorthands $\mathsf{pk}_x \equiv \mathsf{pk}(\mathsf{k}(x))$ and $\mathsf{sk}_x \equiv \mathsf{sk}(\mathsf{k}(x))$. $\langle \_,\_ \rangle$ denotes paring of messages, and $\pi_1(\_)$, $\pi_2(\_)$ denote projections to the first and second components of a pair respectively. □

In addition, there is a set $\mathcal{G}$ of (free) function symbols, containing, for every natural number n, infinitely many symbols whose type is $\mathsf{message}^n \to \mathsf{message}$. The set $\mathcal{N}$ of *names* is an infinite set of symbols that are treated as functions symbols of arity 0 and sort $\mathsf{message}$. $\mathcal{X}$ is an infinite set of *variable symbols*, each coming with a sort $s \in S$. We assume that $\mathcal{F}, \mathcal{G}, \mathcal{N}, \mathcal{X}$ are disjoint.

Terms are built using $\mathcal{F}, \mathcal{G}, \mathcal{N}, \mathcal{X}$, following the sort discipline: for each $s \in S$, let $T_s(\mathcal{F}, \mathcal{G}.\mathcal{N}, \mathcal{X})$ be the smallest set such that

- if $n \in \mathcal{N}$, then $n \in T_{\mathsf{message}}(\mathcal{F}, \mathcal{G}.\mathcal{N}, \mathcal{X})$ and if $x \in \mathcal{X}$ has sort s, then $x \in T_s(\mathcal{F}, \mathcal{G}.\mathcal{N}, \mathcal{X})$
- if $f : s_1 \times \ldots \times s_n \to s$ is a symbol of $\mathcal{F} \cup \mathcal{G}$, and $t_1 \in T_{s_1}(\mathcal{F}, \mathcal{G}.\mathcal{N}, \mathcal{X}), \ldots, t_n \in T_{s_n}(\mathcal{F}, \mathcal{G}.\mathcal{N}, \mathcal{X})$, then $f(t_1, \ldots, t_n) \in T_s(\mathcal{F}, \mathcal{G}.\mathcal{N}, \mathcal{X})$

Note that we do not have implicit coercion: a term of sort bool cannot be seen (also) as a term of sort $\mathsf{message}$. We may however add explicit coercion function symbols.

EXAMPLE 5. With $\mathcal{F}$ as defined in Example 4, using 0-arity symbols $a, a', b \in \mathcal{F}$ as agent names, $r \in \mathcal{N}$ and an additional function symbol g from $\mathcal{G}$,

$$
\begin{array}{l}
\mathsf{if}\ EQ\Big(\pi_1\Big(\mathsf{dec}\big(g(\mathsf{pk}_a, \mathsf{pk}_{a'}, \mathsf{pk}_b), \mathsf{sk}_b\big)\Big), \mathsf{pk}_a\Big) \\
\mathsf{then}\ \Big\{\Big\langle \pi_2\Big(\mathsf{dec}\big(g(\mathsf{pk}_a, \mathsf{pk}_{a'}, \mathsf{pk}_b), \mathsf{sk}_b\big)\Big), n\Big\rangle\Big\}^r_{\mathsf{pk}_a} \\
\mathsf{else}\ \mathbf{0}
\end{array}
$$

is a term of sort $\mathsf{message}$. We will use this term to denote the output of agent b in the protocol in Example 4 (and $g(\mathsf{pk}_a, \mathsf{pk}_{a'}, \mathsf{pk}_b)$ will denote the adversary's input computed from the initial information of public keys $\mathsf{pk}_a, \mathsf{pk}_{a'}, \mathsf{pk}_b$). □

In order to display the formulas more concisely, we will write $\mathsf{if}\ s\ \mathsf{then}\ t$ instead of $\mathsf{if}\ s\ \mathsf{then}\ t\ \mathsf{else}\ \mathbf{0}$.

2.2 Formulas

Our atomic formulas are Boolean-valued equivalences between sequences of terms. Formally, we have for every sequence of sorts $s_1, \ldots, s_n$ a *predicate symbol* that takes $2 \times n$ arguments of sort $(s_1 \times \ldots \times s_n)^2$, which we write as $t_1, \ldots, t_n \sim u_1, \ldots, u_n$ (overloading the notations for the predicate symbols with different types). $t_1, \ldots, t_n \sim u_1, \ldots, u_n$ represents the *indistinguishability* of the two sequences of terms $t_1, \ldots, t_n$ and $u_1, \ldots, u_n$.

Some of the terms $t_1, \ldots, t_n$ may have the sort bool, but $\sim$ is our only predicate symbol: in this paper, we only consider properties of the protocol that can be expressed with this equivalence symbol.

Our set of formulas, which will be used both for axioms and security properties are *first-order formulas built on the above atomic formulas*.

Such formulas should not be confused with tests that might be performed in the protocol. These tests will be represented using *functions* taking values in sort bool (as $\mathtt{EQ}$). Since we do not fix the set of such functions, a priori any kind of test can be performed.

EXAMPLE 6. If we let $g \in \mathcal{G}$ be a function symbol that represents the attacker's action, $r \in \mathcal{N}$ be a name and a, a', b be distinct agent names (constant function symbols in $\mathcal{F}$), then

$$
\begin{array}{l}
\mathsf{if}\ \mathtt{EQ}\Big(\pi_1\Big(\ \mathsf{dec}\big(g(\mathsf{pk}_a, \mathsf{pk}_{a'}, \mathsf{pk}_b), \mathsf{sk}_b\big)\ \Big), \mathsf{pk}_a\Big) \\
\mathsf{then}\ \Big\{\Big\langle \pi_2\Big(\ \mathsf{dec}\big(g(\mathsf{pk}_a, \mathsf{pk}_{a'}, \mathsf{pk}_b), \mathsf{sk}_b\big)\ \Big), n\Big\rangle\Big\}^r_{\mathsf{pk}_a} \\
\sim \\
\mathsf{if}\ \mathtt{EQ}\Big(\pi_1\Big(\ \mathsf{dec}\big(g(\mathsf{pk}_a, \mathsf{pk}_{a'}, \mathsf{pk}_b), \mathsf{sk}_b\big)\ \Big), \mathsf{pk}_{a'}\Big) \\
\mathsf{then}\ \Big\{\Big\langle \pi_2\Big(\ \mathsf{dec}\big(g(\mathsf{pk}_a, \mathsf{pk}_{a'}, \mathsf{pk}_b), \mathsf{sk}_b\big)\ \Big), n\Big\rangle\Big\}^r_{\mathsf{pk}_{a'}}
\end{array}
$$

is an atomic formula without free variable, which expresses the desired privacy property for one session of the private authentication protocol of Example 4. This property states that the adversary cannot tell whether b is checking for the key of a or a'. □

3. SEMANTICS

3.1 First-order interpretation

The models of our formulas are classical first-order models. The algebras are multi-sorted, following our sort discipline.

EXAMPLE 7. Let us consider the formula in Example 6. We illustrate two possible models of our logic, one that is a model and another that is a counter-model of the formula.

- Consider the interpretation domain of terms, constructed on the encryption symbol, pairing, names and keys, as well as

a single error message. The dec symbol is interpreted in the model in such a way that $\mathsf{dec}(\{x\}^z_{\mathsf{pk}_y}, \mathsf{sk}_y) = x$ (here $=$ is the identity on the domain of the model), and decryption returns an error message in all other cases (i.e., in case the argument is not an encryption or the decryption key does not match). The terms are evaluated in a strict way (call by value) and any function applied to an error returns the error. bool is interpreted as a two element set: the interpretation of true and the interpretation of false. $\mathtt{EQ}$ is interpreted as the term identity. $\mathsf{if\_then\_else\_}$ is interpreted as usual. The equivalence is interpreted as static equivalence (as defined in [1]). Suppose we chose different a, a', and b. Now, if we set $g(\mathsf{pk}_a, \mathsf{pk}_{a'}, \mathsf{pk}_b)$ to $\{\langle \mathsf{pk}_b, n_0\rangle\}^{r_0}_{\mathsf{pk}_b}$, (for some $r_0 \in \mathcal{F}$) we get a model of the equivalence, as both members of the equivalence consist in an empty message. If we set $g(\mathsf{pk}_a, \mathsf{pk}_{a'}, \mathsf{pk}_b)$ to $\{\langle \mathsf{pk}_a, n_0\rangle\}^{r_0}_{\mathsf{pk}_b}$, we get a counter model since the first message is evaluated to $\{\langle n_0, n\rangle\}^{r}_{\mathsf{pk}_a}$ while the second is the empty message.

- We may consider exotic models also: Let the domain be the set of two elements $\{0, 1\}$, let pk_a be interpreted as 1, $\mathsf{pk}_{a'}$ is interpreted as 0 and all function symbols return 1, except $\mathtt{EQ}$, which is interpreted as equality, $\mathbf{0}$, which is interpreted as 0 and $\mathsf{if\_then\_else\_}$, which is interpreted as usual. The equivalence is interpreted as equality. Then the first term in the example is interpreted as 1 and the second one is interpreted as 0. The equivalence fails in this model. □

3.2 Computational interpretation

A computational model $\mathcal{M}^c$ is a particular first-order model, which defines both the interpretations of the primitives (function symbols in $\mathcal{F}$) and the attacker's actions (function symbols in $\mathcal{G}$). Such a model is defined as follows:

1. The domain of sort $\mathsf{message}$ is a family of PTIME computable bit string valued random variables indexed by the security parameter. More precisely, the domain D_{message} (or D_{m} in short) is the set of deterministic Turing machines $\mathcal{A}$ equipped with an input (and working) tape and two extra tapes (that will be used for the random inputs). All tapes carry bit strings only, the additional tapes contain infinitely long randomly generated bit strings. We require that the computation time of $\mathcal{A}$ is polynomial in the worst case w.r.t the input (not the content of the extra tapes). One of the extra tapes is used for the honestly generated random values (that are available to the attacker only if they are disclosed, directly or indirectly), while the other is used by the attacker when (s)he draws random values. The length of the machine input is the security parameter. We write $\mathcal{A}(w; \rho_1; \rho_2)$ for the output of the machine $\mathcal{A}$ on input w with extra tape contents ρ_1, ρ_2.

The domain of sort bool is the set of such machines whose output is in $\{0, 1\}$. We denote this by D_{bool} (or D_{b} in short).

2. A function symbol $f \in \mathcal{F}$ (resp. a function in $\mathcal{G}$), $f : s_1 \times \ldots \times s_n \to s$ is interpreted as a mapping $[\![f]\!] : D_{s_1} \times \ldots \times D_{s_n} \to D_s$ that it is defined by some polynomial time (deterministic) Turing machine $\mathcal{A}_f$ as follows: Let $(d_1, \ldots, d_n) \in D_{s_1} \times \ldots \times D_{s_n}$, that is, Turing machines as defined above.

- If $f \in \mathcal{F}$, then $[\![f]\!](d_1, \ldots, d_n)$ is the machine, such that, on input w and extra tapes ρ_1, ρ_2,

$$[\![f]\!](d_1, \ldots, d_n)(w; \rho_1; \rho_2) := \mathcal{A}_f(d_1(w; \rho_1; \rho_2), \ldots, d_n(w; \rho_1; \rho_2))$$

  In other words, we compose the machine $\mathcal{A}_f$ with the machines $d_1, \ldots, d_n$.

  Note that the machine $\mathcal{A}_f$ cannot use directly the tapes ρ_1, ρ_2 (only through $d_1, \ldots, d_n$). Intuitively, the random inputs to the functions in $\mathcal{F}$ must be explicitly given as arguments. For instance, the encryption symbol has three arguments, including the plaintext, the key and the random seed.

- If $g \in \mathcal{G}$, $[\![g]\!](d_1, \ldots, d_n)$ is the machine such that, on input w and extra tapes ρ_1, ρ_2,

$$[\![g]\!](d_1, \ldots, d_n)(w; \rho_1; \rho_2) := \mathcal{A}_g(d_1(w; \rho_1; \rho_2), \ldots, d_n(w; \rho_1; \rho_2); \rho_2)$$

  Note that the machine $\mathcal{A}_g$ cannot use directly the tape ρ_1 (only through $d_1, \ldots, d_n$). Intuitively, the interpretation of function symbols in $\mathcal{G}$ is chosen by the attacker: they cannot use directly the (supposedly) secret values but may use extra random choices from ρ_2. Note, $\mathcal{A}_{g_1}$ and $\mathcal{A}_{g_2}$ for some g_1 and g_2 may use the same random values (that is, the adversary is allowed to reuse the same random input). In what follows, the interpretation of $g_i \in \mathcal{G}$ computes the attacker's i'th input (the attacker is active) when applied on the previous outputs (the attacker is adaptive).

- For all computational models, we require fixed interpretations of the following function symbols:
  - $[\![\mathsf{true}]\!]$ the random variable whose value is 1 on all inputs. Note, $[\![\mathsf{true}]\!]$ is in D_{b}.
  - $[\![\mathsf{false}]\!]$ is the random variable whose value is 0 on all inputs. Note, $[\![\mathsf{false}]\!]$ is in D_{b}.
  - $\mathsf{if\_then\_else\_}$ is interpreted as a function

$$[\![\mathsf{if\_then\_else\_}]\!] : D_{\mathsf{b}} \times D_{\mathsf{m}} \times D_{\mathsf{m}} \to D_{\mathsf{m}}$$

    such that on the triple $(d, d_1, d_2) \in D_{\mathsf{b}} \times D_{\mathsf{m}} \times D_{\mathsf{m}}$, it gives the random variable $[\![\mathsf{if\_then\_else\_}]\!](d, d_1, d_2)$ with

$$[\![\mathsf{if\_then\_else\_}]\!](d, d_1, d_2)(w; \rho_1; \rho_2) := \begin{cases} d_1(w; \rho_1; \rho_2) & \text{if } d(w; \rho_1; \rho_2) = 1 \\ d_2(w; \rho_1; \rho_2) & \text{if } d(w; \rho_1; \rho_2) = 0 \end{cases}$$

  - $\mathtt{EQ}$ is interpreted as the function

$$[\![\mathtt{EQ}]\!] : D_{\mathsf{m}} \times D_{\mathsf{m}} \to D_{\mathsf{b}}$$

    such that $[\![\mathtt{EQ}]\!](d_1, d_2)$ is the random variable for which

$$[\![\mathtt{EQ}]\!](d_1, d_2)(w; \rho_1; \rho_2) := \begin{cases} 1 & \text{if } d_1(w; \rho_1; \rho_2) = d_2(w; \rho_1; \rho_2) \\ 0 & \text{if } d_1(w; \rho_1; \rho_2) \neq d_2(w; \rho_1; \rho_2) \end{cases}$$

  - Finally, $\mathtt{EQL}$ is interpreted as the function $[\![\mathtt{EQL}]\!] : D_{\mathsf{m}} \times D_{\mathsf{m}} \to D_{\mathsf{b}}$ such that $[\![\mathtt{EQL}]\!](d_1, d_2)$ is the random variable for which $[\![\mathtt{EQL}]\!](d_1, d_2)(w; \rho_1; \rho_2) := 1$ if the lengths (as bit strings) of $d_1(w; \rho_1; \rho_2)$ and of $d_2(w; \rho_1; \rho_2)$ are equal, and $[\![\mathtt{EQL}]\!](d_1, d_2)(w; \rho_1; \rho_2) := 0$ if the lengths of $d_1(w; \rho_1; \rho_2)$ and of $d_2(w; \rho_1; \rho_2)$ are different.

3. A name $n \in \mathcal{N}$ is interpreted as the machine $[\![n]\!] = \mathcal{A}_n$ that, given a word of length η, extracts a word of length η from ρ_1. Different names should extract disjoint parts of ρ_1. This machine does not use ρ_2.

This way, all names are drawn uniformly at random from $\{0, 1\}^\eta$; other PTIME computable distribution can be represented using additional function symbols, if necessary.

4. Given a term t, an assignment σ of the free variables of t, taking values in the corresponding domains D_s, a security parameter η and a sample ρ (ρ is a pair $\rho_1; \rho_2$), $[\![t]\!]^{\sigma}_{\eta,\rho}$ is defined recursively as:

- for a variable x, $[\![x]\!]^{\sigma}_{\eta,\rho} := (x\sigma)(1^\eta; \rho)$ (the value of the random variable $x\sigma$ on $1^\eta; \rho$ or the output of the machine interpreting x, with the random tapes ρ on the input 1^η),
- for a name n, $[\![n]\!]^{\sigma}_{\eta,\rho}$ is the output of the machine $\mathcal{A}_n$ on 1^η with tape ρ (which is sometimes written $\rho(n)$),
- for a function symbol $f \in \mathcal{F}$, $[\![f(t_1, \ldots, t_n)]\!]^{\sigma}_{\eta,\rho} := [\![f]\!]([\![t_1]\!]^{\sigma}_{\eta,\rho}, \ldots, [\![t_n]\!]^{\sigma}_{\eta,\rho})$.
- for a function symbol $g \in \mathcal{G}$, $[\![g(t_1, \ldots, t_n)]\!]^{\sigma}_{\eta,\rho} := [\![g]\!]([\![t_1]\!]^{\sigma}_{\eta,\rho}, \ldots, [\![t_n]\!]^{\sigma}_{\eta,\rho}, \rho_2)$.

5. The indistinguishability predicate $\sim$ is interpreted as computational indistinguishability $\approx$ of sequences of elements in D of the same length. That is: $d_1, \ldots, d_n \approx d'_1, \ldots, d'_n$ iff for any deterministic polynomial time Turing machine $\mathcal{A}$,

$$|\textbf{Prob}\{\rho : \mathcal{A}(d_1(1^\eta; \rho), \ldots, d_n(1^\eta; \rho); \rho_2) = 1\} - \textbf{Prob}\{\rho : \mathcal{A}(d'_1(1^\eta; \rho), \ldots, d'_n(1^\eta; \rho); \rho_2) = 1\}|$$

is negligible in η. In particular, given an assignment σ of free variables in D_s, and an interpretation $[\![\cdot]\!]$ of the function symbols as above, $\sim$ is interpreted as the relation $\approx$ between sequences of the same length, which is defined as follows: $[\![t_1, \ldots, t_n]\!] \approx [\![u_1, \ldots, u_n]\!]$ iff for any deterministic polynomial time Turing machine $\mathcal{A}$

$$|\textbf{Prob}\{\rho : \mathcal{A}([\![t_1]\!]^{\sigma}_{\eta,\rho}, \ldots, [\![t_n]\!]^{\sigma}_{\eta,\rho}; \rho_2) = 1\} - \textbf{Prob}\{\rho : \mathcal{A}([\![u_1]\!]^{\sigma}_{\eta,\rho}, \ldots, [\![u_n]\!]^{\sigma}_{\eta,\rho}; \rho_2) = 1\}|$$

is negligible in η.[3] We write $\mathcal{M}^c, \sigma \models t_1 \ldots t_n \sim u_1 \ldots u_n$ in this case, and say that $\mathcal{M}^c, \sigma$ satisfies $t_1 \ldots t_n \sim u_1 \ldots u_n$. Satisfaction of compound formulas is defined from satisfaction of atomic formulas as usual in first-order logic. We write $\mathcal{M}^c, \sigma \models \theta$ if $\mathcal{M}^c, \sigma$ satisfies the first-order formula θ in the above sense. If $\vec{x}$ is the list of free variables in θ, then $\mathcal{M}^c \models \theta$ stands for $\mathcal{M}^c \models \forall \vec{x} \theta$. A formula is *computationally valid* if it is satisfied in all computational models.

REMARK 1. Equality (up to negligible probability) between x and y can be expressed in the computational model by a formula $\mathrm{EQ}(x, y) \sim \mathsf{true}$. □

EXAMPLE 8. If n, n' are two names, then $n \sim n'$ is computationally valid. We will see also in Section 7 other examples of computationally valid formulas.

On the other hand, the following formula is *not* computationally valid:

$$x \sim z \wedge y \sim z \rightarrow \mathsf{if}\ b\ \mathsf{then}\ x\ \mathsf{else}\ y \sim z$$

Consider the random variables b_1, b_2 that are drawn uniformly from $\{0, 1\}$. Then $\mathsf{if}\ b_1\ \mathsf{then}\ b_1\ \mathsf{else}\ b_2$ equals 0 with probability $\frac{1}{4}$ and 1 with probability $\frac{3}{4}$. It is therefore not indistinguishable from b_2, while $b_1 \sim b_2$ and $b_2 \sim b_2$.

□

[3]This difference is called *advantage* of $\mathcal{A}$.

4. PROTOCOLS

We choose to treat protocols as abstract transition systems. We do not commit to any particular way to specify protocols. They could be specified for instance in the applied pi-calculus [1] or any other process calculus. As mentioned in the Introduction, we assume here a bounded number of sessions: each protocol comes with an arbitrary but fixed bound on the number of steps in its execution. We could of course define the protocols and their various semantics without such a bound, but we chose to keep it simple: anyway, our soundness result (just as that of [6]) holds only for computational adversaries that exploit bounded number of sessions in the security parameter.

4.1 The transition system

A protocol is a transition system defined by

- A finite set of states Q together with an ordering $>$, a distinguished initial state q_0 and a set $Q_f \subseteq Q$ of final states.
- For each state $q \in Q$, a linearly ordered (finite) set $T(q)$ of transition rules

  $$q, \{x_1, \ldots, x_n\} \xrightarrow{\theta} (q', s, \{x_1, \ldots, x_n, x\})$$

  where
  - $x_1, \ldots, x_n, x$ are variables.
  - θ is a term of sort bool with variables in $x_1, \ldots, x_n, x$
  - $q, q' \in Q$ are such that $q > q'$.
  - s is a term with variables in $x_1, \ldots, x_n, x$.

  $T(q)$ is empty if and only if $q \in Q_f$. Otherwise, $T(q)$ contains a maximal transition, whose guard θ is true.
- An initial knowledge ϕ_0

Intuitively, the ordering on states ensures the termination and the ordering on transitions specifies in which order the guards θ have to be tried. The ordering on transitions will therefore exclude protocols that use non-deterministic choices. We also chose to assume a maximal transition whose guard is true. This is always possible without loss of generality; it amounts to state explicitly what happens when none of the guards is satisfied.

EXAMPLE 9. Consider the following modification of the protocol in Example 4:

1. $a' \rightarrow b$: $\{\mathsf{pk}_{a'}, n_0\}^{r_0}_{\mathsf{pk}_b}$
2. $b \xrightarrow{\text{if } \mathsf{pk}_a = \mathsf{pk}_{a'}} a'$: $\{n_0, n\}^{r}_{\mathsf{pk}_a}$
   $b \xrightarrow{\text{if } \mathsf{pk}_a \neq \mathsf{pk}_{a'}} a'$: $\{n, n\}^{r}_{\mathsf{pk}_a}$

This protocol can be specified in the applied π-calculus by the two processes

$$A(a', b) = \nu n_0, r_0.\mathsf{out}(\{\langle \mathsf{pk}_{a'}, n_0\rangle\}^{r_0}_{\mathsf{pk}_b}) \mathsf{in}(x).\ \mathsf{let}\ y = \pi_1(\mathsf{dec}(x, \mathsf{sk}_{a'}))\ \mathsf{in}\ \mathsf{if}\ \mathrm{EQ}(y, n_0)\ \mathsf{then}\ \mathrm{OK}$$

$$B(a, b) = \nu n, r.\mathsf{in}(x').\ \mathsf{let}\ y' = \mathsf{dec}(x', \mathsf{sk}_b)\ \mathsf{in}\ \mathsf{if}\ \mathrm{EQ}(\pi_1(y'), \mathsf{pk}_a)\ \mathsf{then}\ \mathsf{out}(\{\langle \pi_2(y'), n\rangle\}^{r}_{\mathsf{pk}_a})\ \mathsf{else}\ \mathsf{out}(\{\langle n, n\rangle\}^{r}_{\mathsf{pk}_a}).$$

OK is an event that can be used for authentication properties, but that will be ignored in the present paper, in which we only consider the privacy property.

One session of this protocol can then be described using the initial knowledge

$$\phi_0 = a, b, a', \mathsf{pk}_a, \mathsf{pk}_b, \mathsf{pk}_{a'}, \{\langle \mathsf{pk}_{a'}, n\rangle\}^r_{\mathsf{pk}_b}.$$

and the following transition system

$$t_1\colon\ q_0 \xrightarrow{\mathrm{EQ}(\pi_1(\mathsf{dec}(x',\mathsf{sk}_b)),\mathsf{pk}_a)} q_1, \{\langle \pi_2(\mathsf{dec}(x',\mathsf{sk}_b)), n'\rangle\}^{r'}_{\mathsf{pk}_a}, x'$$

$$t_2\colon\ q_0 \xrightarrow{\mathsf{true}} q_2, \{\langle n', n'\rangle\}^{r'}_{\mathsf{pk}_a}), x'$$

in which $q_0 > q_1$ and $t_1 > t_2$.

In the full description of one session of the protocol we should add a first step, $q_{-1} \xrightarrow{\mathsf{true}} \{\mathsf{pk}_{a'}, n_0\}^{r_0}_{\mathsf{pk}_b}$ and do not include this message in the initial knowledge of the attacker. We chose the above version, in order to keep the protocol short in some of the next examples. □

Given a protocol P, we associate with each $n \in \mathbb{N}$ a function symbol $g_n \in \mathcal{G}$, whose arity is $n + |\phi_0|$: intuitively such a function symbol will account for the attacker's actions and take as arguments the initial knowledge of the attacker, as well as the n first outputs of the protocol and return a message that will be given as the next input to the protocol.

4.2 Execution and indistinguishability in a model $\mathcal{M}$

Given a first-order structure $\mathcal{M}$, in which all function symbols of $\mathcal{F}, \mathcal{G}$ are interpreted, we let $[\![t]\!]^\sigma_{\mathcal{M}}$ be the interpretation of t in $\mathcal{M}$, given the assignment σ of the free variables of t in the domain of $\mathcal{M}$. Since we only consider a finite set of states, we assume w.l.o.g. that all honestly generated names are generated at once, before the protocol starts.

An *execution of the protocol in* $\mathcal{M}$ is a sequence

$$(q_0, \sigma_0, \phi_0) \to \cdots \to (q_n, \sigma_n, \phi_n)$$

such that σ_0 is empty and, for every $i < n$, there is a transition

$$t_i :\quad q_i, \{x_1, \ldots, x_i\} \xrightarrow{\theta_i} (q_{i+1}, s_{i+1}, \{x_1, \ldots, x_i, x_{i+1}\})$$

such that:

- $\sigma_{i+1} = \sigma_i \uplus \{x_{i+1} \mapsto g_i(\phi_i)\}$ and $[\![\theta_i]\!]^{\sigma_{i+1}}_{\mathcal{M}} = [\![\mathsf{true}]\!]_{\mathcal{M}}$.
- $[\![\theta]\!]^{\sigma_{i+1}}_{\mathcal{M}} \neq [\![\mathsf{true}]\!]_{\mathcal{M}}$ for every t: $q_i, \{x_1, \ldots, x_n\} \xrightarrow{\theta} \ldots$ transition with $t < t_i$.
- $\phi_{i+1} \equiv \phi_i, s_{i+1}\sigma_{i+1}$.

and the state q_n is a final state.

Note that for any protocol P and any first order structure $\mathcal{M}$ there is one and only one execution of the protocol P in $\mathcal{M}$. Intuitively, this follows from the fact that $\mathcal{M}$ determines in particular the attacker's actions, hence the assignments σ_i. Note also that the number of elements in Q bounds the number of execution steps.

If $\mathcal{M}$ also interprets predicate symbol $\sim$, then we can have the following definition.

DEFINITION 1. *Given $\mathcal{M}$, two protocols P and P' are equivalent in $\mathcal{M}$, which we write $P \sim_{\mathcal{M}} P'$ if the two executions of P and P' in $\mathcal{M}$ respectively have the same length and yield respectively (q_n, σ_n, ϕ_n) and $(q'_n, \sigma'_n, \phi'_n)$ such that q_n, q'_n are both final in $\mathcal{M}$ and $\mathcal{M} \models \phi_n \sim \phi'_n$.*

DEFINITION 2. *Given a set of axioms $\mathcal{A}$, two protocols P, P' are* symbolically equivalent *if for every model $\mathcal{M}$ satisfying $\mathcal{A}$, $P \sim_{\mathcal{M}} P'$.*

4.3 Computational execution

A *computational trace* of a protocol P is determined by:

- – an interpretation of symbols in $\mathcal{F}$
  – an attacker $\mathcal{B}$

  } A computational model $\mathcal{M}^c$
- a security parameter η
- random tapes $\rho = \rho_1; \rho_2$

Informally, the attacker will implement both the function symbols in $\mathcal{G}$, i.e., the functions g_i at each step of the protocol, and the distinguisher between the final frames.

Given $P, \mathcal{B}, \eta, \rho$ and the interpretation of the symbols of $\mathcal{F}$ in D_{m}, a computational trace is a sequence:

$$(q_0, \sigma_0, \phi_0) \to \cdots \to (q_n, \sigma_n, \phi_n)$$

such that,

- for every $i < n$, there is a transition

  $$t_i :\quad q_i, \{x_1, \ldots, x_i\} \xrightarrow{\theta_i} (q_{i+1}, s_{i+1}, \{x_1, \ldots, x_i, x_{i+1}\})$$

  such that:

  – $\sigma_{i+1} = \sigma_i \uplus \{x_{i+1} \mapsto \mathcal{B}([\![\phi_i]\!]_{\rho,\eta}; \rho_2)\}$ and $[\![\theta_i]\!]^{\sigma_{i+1}}_{\rho,\eta} = 1$

  – $[\![\theta]\!]^{\sigma_{i+1}}_{\rho,\eta} \neq 1$ for every $t : q_i, \{x_1, \ldots, x_i\} \xrightarrow{\theta} \cdots$ transition with $t < t_i$

  – $\phi_{i+1} = \phi_i, [\![s_{i+1}]\!]^{\sigma_{i+1}}_{\rho,\eta}$.
- q_n is a final state

We write $\Phi(P, \mathcal{B}, \eta, \rho)$ for the final frame ϕ_n in such a trace.

A computational trace can be seen as a particular case of execution in a model of Section 4.2, as $(\mathcal{M}^c, \eta, \rho)$ is a first-order model interpreting the function symbols, but not $\sim$.[4]

A *computational execution* of protocol P is defined by an interpretation of $\mathcal{F}$ and an attacker $\mathcal{B}$ and consists of all computational traces for all values of the security parameter and the random tapes as the input $(1^\eta; \rho)$ runs through all possible values.

The attacker $\mathcal{B}$ determines an interpretation of the function symbols in $\mathcal{G}$, which is the computation of $\mathcal{B}$ in each round. Hence we obtain a model $\mathcal{M}^c$.

We state now what it means for a PPT machine to distinguish two protocols:

DEFINITION 3. *Given two protocols P and P', an interpretation of the function symbols in $\mathcal{F}$, and an attacker $\mathcal{B}$ (which, altogether, defines a computational model $\mathcal{M}^c$),* $\mathcal{B}$ distinguishes *P and P' if:*

$$\mathbf{Prob}\{\rho : \mathcal{B}(\Phi(P, \mathcal{B}, \eta, \rho); \rho_2) = 1\} -$$
$$\mathbf{Prob}\{\rho : \mathcal{B}(\Phi(P', \mathcal{B}, \eta, \rho); \rho_2) = 1\}$$

is non-negligible in η.
We also say that P and P' are distinguishable in the model $\mathcal{M}^c$.

We obtain again the usual notion of computational indistinguishability of protocols, letting $\mathcal{M}^c$ vary, for a fixed interpretation of $\mathcal{F}$: given an interpretation I of the symbols in $\mathcal{F}$, two protocols P and P' are *computationally indistinguishable* if they are indistinguishable in any computational model $\mathcal{M}^c$ extending I, that is, if for any PPT machine $\mathcal{B}$, $\mathcal{B}$ does not distinguish P and P'.

[4]Note that $[\![t]\!]^\sigma_{\mathcal{M}^c}$ (with the notation of Section 4.2) and $[\![t]\!]^\sigma_{\eta,\rho}$ are not the same because $\mathcal{M}^c$ does not fix η and ρ. We omitted $\mathcal{M}^c$ from $[\![t]\!]^\sigma_{\eta,\rho}$ for simplicity, but in fact $[\![t]\!]^\sigma_{\eta,\rho} = [\![t]\!]^\sigma_{\mathcal{M}^c}(1^\eta; \rho)$.

EXAMPLE 10. Let us come back to Example 2. The protocol P has only one transition $q_0 \xrightarrow{\mathsf{true}} q_1, n', x$. The protocol P' has two transitions $t_1 : q_0 \xrightarrow{\mathrm{EQ}(b,0)} q_1, 1 \cdot n, x$ and $t_2 : q_0 \xrightarrow{\mathsf{true}} q_2, 0 \cdot n, x$ with $t_1 > t_2$. Assume we interpret $0, 1$ as $0, 1$ and $\cdot$ as the concatenation. If $[\![n]\!]_\rho = w$ and $[\![b]\!]_\rho = 1$, then the computational execution of P' in the model defined by an attacker $\mathcal{B}$ and the sample ρ is $(q_0, \emptyset, \emptyset) \to (q_2, \sigma, 0w)$, where σ is the attacker's input, which is, here, irrelevant. The other execution (when $[\![b]\!]_\rho = 0$) yields $(q_1, \sigma, 1w)$.

The two processes are computationally indistinguishable: indistinguishability is defined as a global property (for all samples ρ), and the distributions of the outputs for P and P' are identical. □

5. FOLDING THE PROTOCOLS

We claim that it may be assumed w.l.o.g. that, for any control state q, there is at most one transition departing from q. From any protocol P, we construct a protocol $\mathsf{fold}(P)$ that satisfies this property, and which is computationally equivalent to P. They are also equivalent with respect to any model $\mathcal{M}$ in which bool is interpreted as a two element set. Informally, if we see P as a tree, whose nodes are labeled with the sates and the output terms, and edges are labeled with the conditions, then in $\mathsf{fold}(P)$ all states occurring at the same depth d are gathered together and there is a case distinction in the output term: for every path of depth d, the term gathers together the conditions along this path and at the end of the conditions contains (as a subterm) the term labeling the target of this path.

EXAMPLE 11. Let P be defined by the transition rules

$t_0^0 : q_0 \xrightarrow{\theta_0^1} q_1^1, s_1^1, x_1, \qquad t_0^1 : q_0 \xrightarrow{\theta_0^2} q_1^2, s_1^2, x_1,$

$t_0^2 : q_0 \xrightarrow{\mathsf{true}} q_2, \mathbf{0}, x_1,$

$t_1^0 : q_1^1, x_1 \xrightarrow{\theta_1^1} q_2, s_2^1, x_1, x_2, \qquad t_1^1 : q_1^1, x_1 \xrightarrow{\theta_1^2} q_2, s_2^2, x_1, x_2,$

$t_1^2 : q_1^1, x_1 \xrightarrow{\mathsf{true}} q_2, \mathbf{0}, x_1, x_2,$

$t_2^0 : q_1^2, x_1 \xrightarrow{\theta_1^3} q_2, s_2^3, x_1, x_2, \qquad t_2^1 : q_1^2, x_1 \xrightarrow{\mathsf{true}} q_2, s_2^4, x_1, x_2.$

The ordering on the transition is given by $t_i^j > t_i^k$ if $j < k$.

Then $\mathsf{fold}(P)$ contains two transition rules:

$$\{q_0\} \xrightarrow{\mathsf{true}} \mathsf{if}\ \theta_0^1\ \mathsf{then}\ s_1^1\ \mathsf{else}\ (\mathsf{if}\ \theta_0^2\ \mathsf{then}\ s_1^2\ \mathsf{else}\ \mathbf{0}\ ),\ \{q_1^1, q_1^2\}, x_1$$

$$\{q_1^1, q_1^2\} \xrightarrow{\mathsf{true}} \mathsf{if}\ \theta_0^1\ \mathsf{then\ if}\ \theta_1^1\ \mathsf{then}\ s_2^1\ \mathsf{else}\ (\ \mathsf{if}\ \theta_1^2\ \mathsf{then}\ s_2^2\ \mathsf{else}\ \mathbf{0}\ )\ \mathsf{else\ if}\ \theta_0^2\ \mathsf{then}\ (\ \mathsf{if}\ \theta_1^3\ \mathsf{then}\ s_2^3\ \mathsf{else}\ \mathbf{0}\ )\ \mathsf{else}\ s_2^4\ ,\ \{q_2\}, x_1, x_2$$

□

Formally, $\mathsf{fold}(P)$ is constructed as follows. If T is a sequence of transition rules, we define $\mathsf{fold}(T)$ by induction on the number of transition rules in T:

- If T is empty, then $\mathsf{fold}(T) := \mathbf{0}$
- If T is has one rule, *i.e.* T is of the form $q, \vec{x} \xrightarrow{\mathsf{true}} q', s', \vec{x'}$, then $\mathsf{fold}(T) := q', s'$
- If $T = \{q, \vec{x} \xrightarrow{\theta} q', s', \vec{x'}\} \cdot T'$, then

$$\mathsf{fold}(T) := \mathsf{if}\ \theta\ \mathsf{then}\ q', s'\ \mathsf{else}\ \mathsf{fold}(T')$$

We let $Q_0 := \{q_0\}$ and $e_0 := q_0, s_0$, where $s_0 := \phi_0$. For every i, We define inductively e_{i+1} as the expression that we obtain from e_i by replacing every pair of the form q, s occurring in e_i with $\mathsf{fold}(T(q))$, where $T(q)$ is the sequence of transition rules departing from q, in decreasing order.

The protocol P_0 has an initial state Q_0, the states are the sets Q_i of state symbols appearing in the expressions e_i, the transitions are

$$Q_i, \{x_1, \ldots, x_i\} \xrightarrow{\top} Q_{i+1}, s_{i+1}, \{x_1, \ldots, x_i, x\}$$

where s_i is the term obtained by removing the states from the expression e_i.

Given a protocol P, we let $\Phi(\mathsf{fold}(P))$ be the sequence of terms $s_0, s_1, \ldots, s_n$. Note that this sequence may contain variables and also includes the control structure of the protocol.

Note that sequences of transitions of P need not to have all the same length: $\mathsf{fold}(T(q)) = \mathbf{0}$ when there is no transition departing from q and $\mathsf{fold}(P)$ has only one sequence of transitions, whose length equals the length of the longest transition sequence in P.

If-then-else basic axioms

$\forall x, y, z_1, z_2.\ z_1, \mathsf{if\ true\ then}\ x\ \mathsf{else}\ y\ , z_2 \sim z_1, x, z_2$

$\forall x, y, z_1, z_2.\ z_1, \mathsf{if\ false\ then}\ x\ \mathsf{else}\ y\ , z_2 \sim z_1, y, z_2$

Figure 1: Basic axioms

PROPOSITION 1. *If $\mathcal{M}$ is a model satisfying the axioms of the Figure 1 and in which the sort* bool *is interpreted as a two element set $\{[\![\mathsf{true}]\!]_\mathcal{M}, [\![\mathsf{false}]\!]_\mathcal{M}\}$, then $P \sim_\mathcal{M} \mathsf{fold}(P)$ for any protocol P. Moreover, P* and $\mathsf{fold}(P)$ *are computationally indistinguishable.*

The proof is straightforward. Note that we need an assumption on the interpretation of bool: if some condition θ is neither evaluated to true, nor to false, then we have some freedom in the evaluation of "$\mathsf{if}\ \theta\ \mathsf{then}\ s\ \mathsf{else}\ t$ " (hence in the results of $\mathsf{fold}(P)$), while the definition of an execution of P will always behave as if the condition were false.

6. COMPUTATIONAL SOUNDNESS

The following result can be seen as an unconditional soundness theorem: to prove the computational indistinguishability of two (finite) protocols, it is sufficient to check the (symbolic) inconsistency of a set of formulas. Conversely, the consistency of the set of formulas implies the existence of a model, which means that there is an *attack* (for some interpretation of the function symbols).

THEOREM 1. *Assume that P, P' are two protocols. Let A be any set of formulas (axioms). If A and $\Phi(\mathsf{fold}(P)) \not\sim \Phi(\mathsf{fold}(P'))$ are inconsistent, then the protocols P and P' are computationally indistinguishable in any computational model $\mathcal{M}^c$ for which $\mathcal{M}^c \models A$.*

PROOF. Suppose P and P' are distinguishable, that is, there is a model $\mathcal{M}^c$ with $\mathcal{M}^c \models A$ and a computational algorithm that

distinguishes the two protocols. We have to construct an abstract model $\mathcal{M}$, for which $\mathcal{M} \models A \wedge \Phi(\mathsf{fold}(P)) \not\sim \Phi(\mathsf{fold}(P'))$. As noted earlier, we can assume without loss of generality that each computational trace has equal length. Let n denote this number. Since, in the folded protocol, the output terms are independent of the random inputs (but the corresponding bit strings depend of course on this input), for each $i \in [1, \ldots, n]$, we let t_i and t'_i be the output terms such that $[\![t_i]\!], [\![t'_i]\!] \in D$ give the elements in the computational domain that correspond to the i'th message in the corresponding protocol execution. t_i and t'_i are exactly the terms occurring in $\Phi(\mathsf{fold}(P))$ and $\Phi(\mathsf{fold}(P'))$ respectively.

We define the domain of the model $\mathcal{M}$ simply to be the domain of the computational interpretation. The function symbols in $\mathcal{F}$ are interpreted in $\mathcal{M}$ exactly as in $\mathcal{M}^c$.

The free function symbols in $\mathcal{G}$ denote the attacker's computation of its messages from the previous outputs of the agents. That is, the attacker's inputs are $g_1(t_1)$, $g_2(t_1, t_2)$, $g_2(t_1, t_2, t_3)$ etc. for protocol P, and $g_1(t'_1)$, $g_2(t'_1, t'_2)$, $g_2(t'_1, t'_2, t'_3)$ for P'; as it is the same attacker on the two sides, so the free symbols are the same. For $i \geq n$, the interpretation of g_i is irrelevant. For $i < n$, g_i is given by the attacker; in the specification of the protocol attacker, there must be a subroutine that takes as input the first i outputs of the protocol agents, and outputs some message to be sent to a protocol agent again. This subroutine is the interpretation of g_i. The predicate $\sim$ is interpreted as $\approx$ defined in section 3.2. $\mathcal{M} \models A$ since $\mathcal{M}^c \models A$. Finally, $\mathcal{M} \models \Phi(\mathsf{fold}(P)) \not\sim \Phi(\mathsf{fold}(P'))$ also holds, as the protocol adversary taking the last outputs of the protocol agents distinguishes the two protocols, meaning $[\![\Phi(\mathsf{fold}(P))]\!] \not\approx [\![\Phi(\mathsf{fold}(P'))]\!]$. Hence $\mathcal{M} \models \Phi(\mathsf{fold}(P)) \not\sim \Phi(\mathsf{fold}(P'))$. □

7. AXIOMS

In this section we list some axioms that we prove to be computationally sound (some of them under some cryptographic assumptions) and that are sufficient for the analysis of the private authentication protocol in the next section.

As we will see, the first axiom in Figure 2 is sound because function symbols are always interpreted as polynomial time algorithms. Axioms `Refl`, `Sym`, `Trans`, `Restr`, $\mathtt{If}_1$, $\mathtt{Eq}_1$ are always computationally sound.

Axiom $\mathtt{Enc}_{\mathrm{CCA1}}$ is sound as long as the encryption symbol $\{\_\}^{\_}_{\_}$ is interpreted as and IND-CCA1 encryption [12], and axiom $\mathtt{Enc}_{\mathrm{KP}}$ is sound as long as the encryption satisfies key privacy [11].

Axioms $\mathtt{Enc}_{\mathrm{CCA1}}$ and $\mathtt{Enc}_{\mathrm{KP}}$ are actually axiom schemes: the symbols: $r, u, u', u'', a...$ represent any terms, $\vec{v}$ is any finite sequence of terms. They are guarded by constraints, that we explain below. An axiom scheme is more precisely of the form

$$\gamma \quad \| \quad \theta(\vec{v})$$

where γ is a constraint and $\vec{v}$ are the logical variables of θ, and it denotes the set of all instances $\theta(\vec{v})\sigma$ where σ assigns closed terms to the logical variables, and $\sigma \models \gamma$. Such a scheme is a recursive set of formulas as long as the constraints are recursive (which is the case in our examples).

$\mathsf{fresh}(\vec{n}, \vec{v})$ is satisfied by a substitution σ if any element of $\vec{n}$ is mapped to a name by σ and none of these names appears in $\vec{v}\sigma$. An assignment σ satisfies the constraint $\vec{\mathsf{sk}} \not\sqsubseteq_{\not\mathrm{d}} \vec{v}$ if secret keys in $\vec{\mathsf{sk}}\sigma$ only occur in $\vec{v}\sigma$ in decryption key position. Finally, $\sigma \models \mathsf{nodec}(\vec{\mathsf{sk}}; \vec{v})$ if the vector of terms $\vec{v}\sigma$ does not contain any decryption with any of the keys in the vector of secret keys $\vec{\mathsf{sk}}\sigma$.

In the axioms below, $b, w, x, y, z, x_1, \ldots, x_n, \ldots$ are variables, that are all implicitly universally quantified. b has sort bool, while the other variables have arbitrary sorts, provided the terms displayed in the axioms are well-typed.

- `Indcong`: for any $f : \mathsf{message}^n \to \mathsf{message}$ $x_1, ..., x_n \sim y_1, ..., y_n \longrightarrow f(x_1, ..., x_n) \sim f(y_1, ..., y_n)$
- `Refl`: $\vec{x} \sim \vec{x}$
- `Sym`: $\vec{x} \sim \vec{y} \longrightarrow \vec{y} \sim \vec{x}$
- `Trans`: $\vec{x} \sim \vec{y} \wedge \vec{y} \sim \vec{z} \longrightarrow \vec{x} \sim \vec{z}$
- `Restr`: If p projects and permutes onto a sublist, then $\vec{x} \sim \vec{y} \longrightarrow p(\vec{x}) \sim p(\vec{y})$.
- $\mathtt{If}_1$:

$$b, \vec{w}, x \sim b, \vec{w}, z \ \wedge \ b, \vec{w}, y \sim b, \vec{w}, z \longrightarrow \vec{w}, \mathsf{if}\ b\ \mathsf{then}\ x\ \mathsf{else}\ y \ \sim \ \vec{w}, z$$

- $\mathtt{Eq}_1$: $\vec{w}, \mathtt{EQ}(x, x) \sim \vec{w}, \mathsf{true}$
- $\mathtt{Enc}_{\mathrm{CCA1}}$ (computationally sound for IND-CCA1 encryptions): For $r, r'; \vec{v}, u, u', u'', a$, if the constraints $\mathsf{fresh}(r, r'; \vec{v}, u, u', u'', \mathsf{pk}_a, \mathsf{sk}_a)$ and $\mathsf{sk}_a \not\sqsubseteq_{\not\mathrm{d}} \vec{v}, u, u', u''$ hold, then

$$\vec{v}, \mathsf{if}\ \mathtt{EQL}(u, u')\ \mathsf{then}\ \{u\}^r_{\mathsf{pk}_a}\ \mathsf{else}\ u'' \sim \vec{v}, \mathsf{if}\ \mathtt{EQL}(u, u')\ \mathsf{then}\ \{u'\}^{r'}_{\mathsf{pk}_a}\ \mathsf{else}\ u''$$

- $\mathtt{Enc}_{\mathrm{KP}}$ (Computationally sound for encryptions satisfying Key Privacy): For $r, r'; \vec{v}, u, a, a'$, if the constraints $\mathsf{fresh}(r, r'; \vec{v}, u, \mathsf{pk}_a, \mathsf{sk}_a, \mathsf{pk}_{a'}, \mathsf{sk}_{a'})$ and $\mathsf{nodec}(\mathsf{sk}_a, \mathsf{sk}_{a'}; \vec{v}, u)$ and $\mathsf{sk}_a, \mathsf{sk}_{a'} \not\sqsubseteq_{\not\mathrm{d}} \vec{v}, u$ hold, then

$$\vec{v}, \{u\}^r_{\mathsf{pk}_a} \sim \vec{v}, \{u\}^{r'}_{\mathsf{pk}_{a'}}$$

Figure 2: Some axioms

PROPOSITION 2. *The axioms* `Indcong`, `Refl`, `Sym`, `Trans`, `Restr`, $\mathtt{If}_1$, $\mathtt{Eq}_1$ *are computationally sound.*

PROOF. For `Indcong`: if σ is an assignment of the variables such that a machine $\mathcal{A}$ breaks the indistinguishability $f(t_1, \ldots, t_n) \sim f(u_1, \ldots, u_n)$, then we consider a machine $\mathcal{B}$, which, on inputs $m_1, \ldots, m_n$, first computes $[\![f]\!](m_1, \ldots, m)$ and then runs $\mathcal{A}$ on the result. $\mathcal{B}$ is still polynomial time, because we assumed that the computational interpretation of function symbols are polynomial time computable. The advantage of $\mathcal{B}$ is the same as the advantage of $\mathcal{A}$.

Axiom `Refl` is obvious: Something cannot be distinguished from itself.

Axiom `Sym` is a consequence of the symmetry of the arguments in the computational definition of indistinguishability.

Axiom `Trans` is a consequence of the triangle inequality: if the advantage of an attacker in breaking $x \sim z$ is larger than p, the very same attacker breaks either $x \sim y$ or $y \sim z$ with advantage at least $\frac{p}{2}$.

Axiom `Restr` is proved the same way as axiom `Indcong`.

For $\mathtt{If}_1$, consider the following. Assume that σ is an assignment of $\vec{w}, b, x, y, z$ to random variables and let $\mathcal{A}$ be an attacker

machine. Let

$$p_1 = \textbf{Prob}\{\rho : \mathcal{A}([\![\vec{w}, \mathsf{if}\ b\ \mathsf{then}\ x\ \mathsf{else}\ y\,]\!]^{\sigma}_{\rho,\eta}; \rho_2) = 1\},$$
$$p_2 = \textbf{Prob}\{\rho : \mathcal{A}([\![\vec{w}, z]\!]^{\sigma}_{\rho,\eta}; \rho_2) = 1\},$$
$$p^x_{1,1} = \textbf{Prob}\{\rho : \mathcal{A}([\![\vec{w}, x]\!]^{\sigma}_{\rho,\eta}; \rho_2) = 1 \ \&\ [\![b]\!]^{\sigma}_{\rho,\eta} = 1\},$$
$$p^y_{1,2} = \textbf{Prob}\{\rho : \mathcal{A}([\![\vec{w}, y]\!]^{\sigma}_{\rho,\eta}; \rho_2) = 1 \ \&\ [\![b]\!]^{\sigma}_{\rho,\eta} = 0\}.$$

By definition of the semantics of if_then_else_, $p_1 = p^x_{1,1} + p^y_{1,2}$. On the other hand, $p_2 = p^z_{1,1} + p^z_{1,2}$.

Let $\mathcal{B}_1$ (resp. $\mathcal{B}_2$) be the machine that, on input m_1, m_2 (and random tape ρ_2) returns $\mathcal{A}(m_2; \rho_2)$ when $m_1 = 1$ (resp. $m_1 = 0$) and flips a coin (use ρ_2) returning either 0 or 1 with equal probability, if $m_1 \neq 1$ (resp. $m_1 \neq 0$). $\textbf{Prob}\{\rho : \mathcal{B}_1([\![b]\!]^{\sigma}_{\rho,\eta}, [\![\vec{w}, x]\!]^{\sigma}_{\rho,\eta}; \rho_2) = 1\} = \frac{1}{4} + p^x_{1,1}$ and $\textbf{Prob}\{\rho : \mathcal{B}_1([\![b]\!]^{\sigma}_{\rho,\eta}, [\![\vec{w}, z]\!]^{\sigma}_{\rho,\eta}; \rho_2) = 1\} = \frac{1}{4} + p^z_{1,1}$. Hence $\mathcal{B}_1$ distinguishes $b, \vec{w}, x$ from $b, \vec{w}, z$ with probability at least $|p^x_{1,1} - p^z_{1,1}|$. Similarly, the machine $\mathcal{B}_2$ distinguishes b, y from b, z with probability at least $|p^y_{1,2} - p^z_{1,2}|$ Since $|p_1 - p_2| \leq |p^x_{1,1} - p^z_{1,1}| + |p^y_{1,2} - p^z_{1,2}|$, one of the machines $\mathcal{B}_i$ succeeds in distinguishing the two pairs of messages with advantage at least $\frac{1}{2}|p_1 - p_2|$: if an attacker can distinguish $\vec{w}, \mathsf{if}\ b\ \mathsf{then}\ x\ \mathsf{else}\ y$ from $\vec{w}, z$ with an advantage ϵ, then either there is a machine that distinguishes $b, \vec{w}, x$ from $b, \vec{w}, z$ with an advantage at least $\frac{\epsilon}{2}$ or there is a machine that distinguishes $b, \vec{w}, y$ from $b, \vec{w}, z$ with an advantage at least $\frac{\epsilon}{2}$.

The soundness of $\mathtt{Eq}_1$ is immediate from the computational interpretation of EQ. □

We do not define IND-CCA1 security here as it is a very standard notion. The precise definition can be found for example in [12]. The idea is, the following: An oracle generates a random bit and a secret key–public key pair, and publishes the public key. Based on this public key, a PPT attacker algorithm submits two messages of its choice to the oracle, and the oracle encrypts one based on its internal bit. Before doing that though, the attacker algorithm is allowed to request decryptions from the oracle with the secret key, but not after. Then, when received the encryption from the oracle, the attacker has to compute a bit. If the oracle's internal bit and the bit computed by the PPT attacker agree with a probability that is never non-negligibly better than $1/2$ for any such attacker, then the encryption is IND-CCA1 secure. The absolute value of difference between the probability that the algorithm computes the correct bit and $1/2$ is called the *advantage* of the attacker.

PROPOSITION 3. *The axiom $\mathtt{Enc}_{\mathrm{CCA1}}$ is computationally sound if the encryption scheme is IND-CCA1 secure.*

PROOF. The proof goes as usual in the literature concerning computational soundness. Let $t \equiv \mathsf{if}\mathtt{EQL}(u, u')\mathsf{then}\{u\}^r_{\mathsf{pk}_a}\mathsf{else}u''$ and let $t' \equiv \mathsf{if}\ \mathtt{EQL}(u, u')\ \mathsf{then}\ \{u'\}^{r'}_{\mathsf{pk}_a}\ \mathsf{else}\ u''$. Assume that there is an instance of the axiom scheme (a substitution that satisfies the constraints), and a computational interpretation, in which $\vec{v}, t \not\sim \vec{v}, t'$ is satisfied. From this, we construct an algorithm winning the IND-CCA1 game against the encryption with pk_a. (Note that $\vec{v}$, u, u' were assumed to be closed, so we do not need σ.)

According to our computational semantics in Section 3.2, $\vec{v}, t \not\sim \vec{v}, t'$ is satisfied in the computational interpretation means that there is a polynomial time Turing machine $\mathcal{A}$ such that

$$|\textbf{Prob}\{\rho = (\rho_1, \rho_2) : \mathcal{A}([\![\vec{v}]\!]_{\eta,\rho}, [\![t]\!]_{\eta,\rho}; \rho_2) = 1\} - \textbf{Prob}\{\rho = (\rho_1, \rho_2) : \mathcal{A}([\![\vec{v}]\!]_{\eta,\rho}, [\![t']\!]_{\eta,\rho}; \rho_2) = 1\}| \quad (1)$$

is not negligible in η.

Then an algorithm $\mathcal{B}$ that wins the IND-CCA1 game against pk_a can be constructed: $\mathcal{B}$ is given $[\![\mathsf{pk}(k)]\!]_{\rho,\eta}$, as well as all interpretations of names, except k. (More precisely, $\mathcal{B}$ can draw himself these names). Then $\mathcal{B}$ computes $[\![\vec{v}]\!]_{\rho}, [\![u]\!]_{\rho}, [\![u']\!]_{\rho}, [\![u'']\!]_{\rho}$: as $\vec{v}$, u, u', u'' contain sk_a in the position of a decryption key only, $\mathcal{B}$ can perform these computations by requesting the decryption oracle when necessary. Then, whenever $[\![u]\!]_{\eta,\rho}$, $[\![u']\!]_{\eta,\rho}$ have the same length, $\mathcal{B}$ submits them to the encryption oracle, which returns either $[\![\{u\}^r_{\mathsf{pk}_a}]\!]_{\eta,\rho}$ or $[\![\{u'\}^{r'}_{\mathsf{pk}_a}]\!]_{\eta,\rho}$ depending on the oracle's internal random bit. Note that the oracle generates fresh r or r', the adversary does not have control over that. Therefore we needed the freshness constraint in the axiom. Let $c(\eta, \rho)$ denote whatever the oracle returns. Then $\mathcal{B}$ runs $\mathcal{A}$ as a subroutine on $\vec{v}$, $c(\eta, \rho)$ if the lengths of $[\![u]\!]_{\eta,\rho}$, $[\![u']\!]_{\eta,\rho}$ are the same, and, if different, then $\mathcal{B}$ runs $\mathcal{A}$ on $[\![\vec{v}]\!]_{\eta,\rho}$, $[\![u'']\!]_{\eta,\rho}$. Finally, $\mathcal{B}$ outputs whatever $\mathcal{A}$ returns as the guess of the oracles internal bit.

Let b denote the oracle's internal bit. $\mathcal{B}$ wins when either the oracle's internal bit is 0 (encrypting $[\![u]\!]_{\eta,\rho}$) and $\mathcal{A}$ returns 0, or when the oracle's internal bit is 1 (encrypting $[\![u']\!]_{\eta,\rho}$) and $\mathcal{A}$ returns 1. Hence, the advantage of $\mathcal{B}$ is only negligibly different from

$$|\textbf{Prob}\{\mathcal{A}([\![\vec{v}]\!]_{\eta,\rho}, [\![t]\!]_{\eta,\rho}; \rho_2) = 0 \wedge b = 0\} + \textbf{Prob}\{\mathcal{A}([\![\vec{v}]\!]_{\eta,\rho}, [\![t']\!]_{\eta,\rho}; \rho_2) = 1 \wedge b = 1\} - 1/2|$$

As the probability of $b = 1$ and $b = 0$ are both $1/2$, with conditional probabilities, the above equals

$$|\textbf{Prob}\{\mathcal{A}([\![\vec{v}]\!]_{\eta,\rho}, [\![t]\!]_{\eta,\rho}; \rho_2) = 0 \mid b = 0\} \cdot 1/2 + \textbf{Prob}\{\mathcal{A}([\![\vec{v}]\!]_{\eta,\rho}, [\![t']\!]_{\eta,\rho}; \rho_2) = 1 \mid b = 1\} \cdot 1/2 - 1/2|.$$

However, substituting

$$\textbf{Prob}\{\mathcal{A}([\![\vec{v}]\!]_{\eta,\rho}, [\![t]\!]_{\eta,\rho}; \rho_2) = 0 \mid b = 0\} = 1 - \textbf{Prob}\{\mathcal{A}([\![\vec{v}]\!]_{\eta,\rho}, [\![t]\!]_{\eta,\rho}; \rho_2) = 1 \mid b = 0\},$$

we obtain

$$|\textbf{Prob}\{\mathcal{A}([\![\vec{v}]\!]_{\eta,\rho}, [\![t]\!]_{\eta,\rho}; \rho_2) = 1 \mid b = 0\} \cdot 1/2 - \textbf{Prob}\{\mathcal{A}([\![\vec{v}]\!]_{\eta,\rho}, [\![t']\!]_{\eta,\rho}; \rho_2) = 1 \mid b = 1\} \cdot 1/2|.$$

This in turn is (as the choice of b and of the other names as well as the algorithm $\mathcal{A}$ are independent)

$$|\textbf{Prob}\{\mathcal{A}([\![\vec{v}]\!]_{\eta,\rho}, [\![t]\!]_{\eta,\rho}; \rho_2) = 1\} \cdot 1/2 - \textbf{Prob}\{\mathcal{A}([\![\vec{v}]\!]_{\eta,\rho}, [\![t']\!]_{\eta,\rho}; \rho_2) = 1\} \cdot 1/2|,$$

which is just the half of expression (1), which was assumed to be non-negligible. Therefore, the IND-CCA1 attacker $\mathcal{B}$ wins with non-negligible probability. □

Note that we did not need to call the decryption oracle after the encryption oracle: we do not need IND-CCA2 for this axiom.

In order to be able to verify our example protocol, we also need the encryption to ensure key privacy. For this, we included the last axiom, which we will see to be sound as long as the encryption satisfies the following notion that appeared in [11].

DEFINITION 4 (KEY PRIVACY). *Let us consider a public-key encryption scheme (that is, computational interpretation of* pk, sk, {}, dec, *such that* $[\![\mathsf{dec}]\!]([\![\{x\}^r_{\mathsf{pk}(k)}]\!]^{\sigma}_{\rho,\eta}, [\![\mathsf{sk}(k)]\!]_{\rho,\eta}) = [\![x]\!]^{\sigma}_{\rho,\eta}$*). This scheme ensures* key privacy *if, for any PPT $\mathcal{B}$ with an oracle,*

$$\textbf{Prob}\{\rho : \mathcal{B}^{O_k}([\![\mathsf{pk}(k)]\!]_{\rho,\eta}, [\![\mathsf{pk}(k')]\!]_{\rho,\eta}; \rho_2) = 1\} - \textbf{Prob}\{\rho : \mathcal{B}^{O_{k'}}([\![\mathsf{pk}(k)]\!]_{\rho,\eta}, [\![\mathsf{pk}(k')]\!]_{\rho,\eta}; \rho_2) = 1\}$$

is negligible. Here O_k is the encryption oracle encrypting with $\mathsf{pk}(k)$: $O_k(x) = [\![\{\_\}\_]\!](x, [\![\mathsf{pk}(k)]\!], s)$, where s is a random seed freshly generated by the oracle.

In other words, key privacy holds, if no algorithm $\mathcal{B}$ can distinguish if the encryption oracle encrypts with key k or with key k'.

PROPOSITION 4. *The axiom* $\texttt{Enc}_{\texttt{KP}}$ *is computationally sound if the encryption scheme satisfies key-privacy.*

PROOF. Assume that there is an instance of the axiom with terms satisfying the constraints and such that $\mathcal{A}$ is a PPT machine and

$$|\mathbf{Prob}\{\rho : \mathcal{A}([\![\vec{v}]\!]_{\rho,\eta}, [\![\{u\}^r_{\mathsf{pk}_a}]\!]_{\rho,\eta}, \rho_2) = 1\}$$
$$-\mathbf{Prob}\{\rho : \mathcal{A}([\![\vec{v}]\!]_{\rho,\eta}, [\![\{u\}^r_{\mathsf{pk}_{a'}}]\!]_{\rho,\eta}, \rho_2) = 1\}|$$

is non negligible. That is, the axiom is broken by an attacker $\mathcal{A}$. An attacker $\mathcal{B}$ on key privacy can compute by himself the computational interpretation of $\vec{v}, u$ and submit $[\![u]\!]_{\rho,\eta}$ to the encryption oracle. Then his advantage on the key privacy game is the same as the advantage of $\mathcal{A}$ in distinguishing the two terms. □

We note that axiomatizing other notions as CPA security or CCA2 security is not particularly difficult either in this framework. We chose the above two because those were what we needed for our example protocol in the next section.

8. EXAMPLE PROTOCOL VERIFICATION

In this section we turn to the protocol designed in [3]. The protocol was derived from an authentication protocol, which did not ensure privacy. The authors of [3] modified that original protocol adding a "decoy" message, and then claimed that the new protocol preserved privacy. This is proved symbolically in [3], and also in [17] using a general (automated) procedure. What we do here is completely different from the existing procedures; we use a first-order refutation (inconsistency) proof, while in spirit, the symbolic proofs rely on an induction over the possible attacker's actions. Our proof allows to identify that there is a need to check the input length: this yields a new protocol. Thanks to Theorem 1, we obtain (to our knowledge the first) proof of the corrected protocol in the computational model. Finally, since the inconsistency is derived from the axioms of Figure 2, the privacy of the protocol only requires the encryption scheme to be IND-CCA1 and to preserve key-privacy.

The original protocol (in our Example 4), which does not preserve privacy contains two processes, but only one is relevant here (just like in Example 9, we include the first message of $A(a', b)$ in the initialization):

$$B(a,b) = \nu n \nu r.\mathsf{in}(x).\ \mathsf{let}\ y = \mathsf{dec}(x, \mathsf{sk}_b)\ \mathsf{in}\ \mathsf{if}\ \pi_1(y) = \mathsf{pk}_a$$
$$\mathsf{then}\ \mathsf{out}(\{\langle \pi_2(y), n\rangle\}^r_{\mathsf{pk}_a}.$$

That is, if the agent b receives an x that decrypts correctly with his decryption key to a pair of the public key of agent a and some bit string, then it returns that bit string encrypted with the public key of a. Anonymity would mean that $B(a,b) \sim B(a',b)$ that is, it cannot be decided if agent b is checking for the key of a or the key of a'. We will show that $B(a,b) \not\sim B(a',b)$.

The corrected version of this protocol is (in Example 9)

$$\begin{aligned} B'(a,b) = \nu n \nu r.\mathsf{in}(x).\quad & \mathsf{let}\ y = \mathsf{dec}(x, \mathsf{sk}_b)\ \mathsf{in} \\ & \mathsf{if}\ \mathtt{EQ}(\pi_1(y), \mathsf{pk}_a) \\ & \mathsf{then}\ \mathsf{out}(\{\langle \pi_2(y), n\rangle\}^r_{\mathsf{pk}_a}) \\ & \mathsf{else}\ \mathsf{out}(\{\langle n, n\rangle\}^r_{\mathsf{pk}_a}). \end{aligned}$$

That is, here b responds with an encryption even if the test fails.

This much correction is sufficient for proving the equivalence of $B'(a,b)$ and $B'(a',b)$ in the symbolic (Dolev-Yao) model, but it is not sufficient to show $B'(a,b) \sim B'(a',b)$ in our model. What is missing is a length test (unless the encryption scheme hides the lengths of the messages); it is easy to construct a computational attack in case this test is not performed. We consider therefore the following third version of the protocol:

$$\begin{aligned} B''(a,b) = \nu n \nu r.\mathsf{in}(x).\mathsf{let}\ y = \mathsf{dec}(x, \mathsf{sk}_b)\ \mathsf{in} \\ \mathsf{if}\ \mathtt{EQ}(\pi_1(y), \mathsf{pk}_a) \\ \quad \mathsf{then\ if}\quad \mathtt{EQL}(\langle \pi_2(y), n\rangle, \langle n, n\rangle) \\ \qquad\qquad \mathsf{then\ out}(\{\langle \pi_2(y), n\rangle\}^r_{\mathsf{pk}_a}) \\ \qquad\qquad \mathsf{else\ out}(\{\langle n, n\rangle\}^r_{\mathsf{pk}_a}) \\ \quad \mathsf{else\ out}(\{\langle n, n\rangle\}^r_{\mathsf{pk}_a}) \end{aligned}$$

The axioms of Section 7 together with $B''(a,b) \not\sim B''(a',b)$ and the agent checks yield an inconsistency, as we show below.

Let us consider the symbolic execution of the first case (case of B): For the protocols $B(a,b)$ and $B(a',b)$, the first published message is $\{\langle \mathsf{pk}_a, n_a\rangle\}^{r_a}_{\mathsf{pk}_b}$ (compared to Example 4, we switched here to a from a' as it does not matter because of the symmetry of the problem), besides the public keys. That is, we let $\phi_0 \equiv \mathsf{pk}_a, \mathsf{pk}_{a'}, \mathsf{pk}_b, \{\langle \mathsf{pk}_a, n_a\rangle\}^{r_a}_{\mathsf{pk}_b}$. Then, the next message sent by the honest agent b is (let $h = \mathsf{dec}(g(\phi_0), \mathsf{sk}_b)$ where g is the attacker's computation):

$$\phi_1^1 \equiv \phi_0, \mathsf{if}\ \mathtt{EQ}(\pi_1(h), \mathsf{pk}_a)\ \mathsf{then}\ \{\langle \pi_2(h), n\rangle\}^r_{\mathsf{pk}_a}\ \mathsf{else}\ \mathbf{0}$$

in case of $B(a,b)$, and

$$\phi_1^{1'} \equiv \phi_0, \mathsf{if}\ \mathtt{EQ}(\pi_1(h), \mathsf{pk}_{a'})\ \mathsf{then}\ \{\langle \pi_2(h), n\rangle\}^r_{\mathsf{pk}_{a'}}\ \mathsf{else}\ \mathbf{0}$$

in case of $B(a',b)$. Anonymity at this point would be $\phi_1^1 \sim \phi_1^{1'}$, so consider its negation $\phi_1^1 \not\sim \phi_1^{1'}$. In our symbolic execution, an attack means that the negation of the security property is consistent with the axioms. We therefore indicate that there is a model satisfying all of them. The model is defined as: Let bool contain only true and false, and let $\mathsf{message}$ contain the public and secret keys, distinct names for random inputs of the encryption and n and terms built on them via pairing and encryption. Let $g(\phi_0) = \{\langle \mathsf{pk}_a, n_a\rangle\}^{r_a}_{\mathsf{pk}_b}$ and $EQ(\mathsf{pk}_a, \mathsf{pk}_{a'}) = \mathsf{false}$, and we take the usual interpretation of ifthenelse and $\mathbf{0} \not\sim \{\langle \pi_2(h), n\rangle\}^r_{\mathsf{pk}_a}$. In other words, the attacker just forwards $\{\langle \mathsf{pk}_a, n_a\rangle\}^{r_a}_{\mathsf{pk}_b}$; in this case $B(a,b)$ responds with an encryption, $\phi_1^1 = \phi_0, \{\langle \pi_2(h), n\rangle\}^r_{\mathsf{pk}_a}$, but for $B(a',b)$ the test fails and $\phi_1^{1'} = \phi_0, \mathbf{0}$. Here, $=$ is the identity in the model. It is not hard to see that the function symbols can be defined on all elements of the domain such that the axioms are satisfied.

In the last case (case of B''), for simplicity, we don't display the full transition system. We have in the symbolic execution again (let $h = \mathsf{dec}(g(\phi_0), \mathsf{sk}_b)$)

$$\phi_1^2 \equiv \phi_0,\ \begin{array}{l} \mathsf{if}\ \mathtt{EQ}(\pi_1(h), \mathsf{pk}_a)\ \mathsf{then\ if}\ \mathtt{EQL}(\langle \pi_2(h), n\rangle, \langle n, n\rangle) \\ \mathsf{then}\{\langle \pi_2(h), n\rangle\}^r_{\mathsf{pk}_a}\ \mathsf{else}\ \{\langle n, n\rangle\}^r_{\mathsf{pk}_a} \\ \mathsf{else}\ \{\langle n, n\rangle\}^r_{\mathsf{pk}_a} \end{array}$$

$$\phi_1^{2'} \equiv \phi_0,\ \begin{array}{l} \mathsf{if}\ \mathtt{EQ}(\pi_1(h), \mathsf{pk}_{a'})\ \mathsf{then\ if}\ \mathtt{EQL}(\langle \pi_2(h), n\rangle, \langle n, n\rangle) \\ \mathsf{then}\{\langle \pi_2(h), n\rangle\}^r_{\mathsf{pk}_{a'}}\ \mathsf{else}\ \{\langle n, n\rangle\}^r_{\mathsf{pk}_{a'}} \\ \mathsf{else}\ \{\langle n, n\rangle\}^r_{\mathsf{pk}_{a'}} \end{array}$$

where ϕ_0 is the same as in Example 9. Negation of the security property is again $\phi_1^2 \not\sim \phi_1^{2'}$. From $\texttt{Enc}_{\texttt{CCA1}}$, we have

$$\begin{array}{l} \mathtt{EQ}(\pi_1(h), \mathsf{pk}_a), \\ \mathsf{if}\ \mathtt{EQL}(\langle \pi_2(h), n\rangle, \langle n, n\rangle)\ \mathsf{then}\ \{\langle \pi_2(h), n\rangle\}^r_{\mathsf{pk}_a}\ \mathsf{else}\ \{\langle n, n\rangle\}^r_{\mathsf{pk}_a} \\ \sim \\ \mathtt{EQ}(\pi_1(h), \mathsf{pk}_a), \\ \mathsf{if}\ \mathtt{EQL}(\langle \pi_2(h), n\rangle, \langle n, n\rangle)\ \mathsf{then}\ \{\langle n, n\rangle\}^r_{\mathsf{pk}_a}\ \mathsf{else}\ \{\langle n, n\rangle\}^r_{\mathsf{pk}_a} \end{array}$$

From $\mathtt{If}_1$ (and $\mathtt{Refl}$ and $\mathtt{Restr}$), we have (with the roles $\vec{w} \equiv \mathtt{EQ}(\pi_1(h), \mathsf{pk}_a)$ and $b \equiv \mathtt{EQL}(\langle \pi_2(h), n\rangle, \langle n, n\rangle)$ and $\vec{x} \equiv \vec{y} \equiv \vec{z} \equiv \{\langle n, n\rangle\}^r_{\mathsf{pk}_a}$):

$$\mathtt{EQ}(\pi_1(h), \mathsf{pk}_a),\ \mathsf{if}\ \mathtt{EQL}(\langle \pi_2(h), n\rangle, \langle n, n\rangle)\ \mathsf{then}\ \{\langle n, n\rangle\}^r_{\mathsf{pk}_a}\ \mathsf{else}\ \{\langle n, n\rangle\}^r_{\mathsf{pk}_a} \sim \mathtt{EQ}(\pi_1(h), \mathsf{pk}_a), \{\langle n, n\rangle\}^r_{\mathsf{pk}_a}$$

Hence, by transitivity,

$$\mathtt{EQ}(\pi_1(h), \mathsf{pk}_a),\ \mathsf{if}\ \mathtt{EQL}(\langle \pi_2(h), n\rangle, \langle n, n\rangle)\ \mathsf{then}\ \{\langle \pi_2(h), n\rangle\}^r_{\mathsf{pk}_a}\ \mathsf{else}\ \{\langle n, n\rangle\}^r_{\mathsf{pk}_a} \sim \mathtt{EQ}(\pi_1(h), \mathsf{pk}_a), \{\langle n, n\rangle\}^r_{\mathsf{pk}_a}.$$

Applying this result on ϕ_1^2, using $\mathtt{If}_1$ (with roles $\vec{w} \equiv \phi_0$ and $b \equiv \mathtt{EQ}(\pi_1(h), \mathsf{pk}_a)$ and $y \equiv x \equiv \{\langle n, n\rangle\}^r_{\mathsf{pk}_a}$ and $x \equiv \mathsf{if}\,\mathtt{EQL}(\langle \pi_2(h), n\rangle, \langle n, n\rangle)\,\mathsf{then}\,\{\langle \pi_2(h), n\rangle\}^r_{\mathsf{pk}_a}\,\mathsf{else}\,\{\langle n, n\rangle\}^r_{\mathsf{pk}_a}$):

$$\phi_1^2 \sim \phi_0, \{\langle n, n\rangle\}^r_{\mathsf{pk}_a}$$

With exactly the same reasoning, we can derive from the axioms: $\phi_1^{2\prime} \sim \phi_0, \{\langle n, n\rangle\}^r_{\mathsf{pk}_{a'}}$ Now, using the axiom $\mathtt{Enc}_{KP}$, we get that $\phi_1^2 \sim \phi_1^{2\prime}$, with which $\phi_1^2 \not\sim \phi_1^{2\prime}$ is inconsistent.

This proves the security of the protocol in any model that satisfies the axioms, in particular any computational model in which the encryption scheme is IND-CCA1 and satisfies key privacy.

9. CONCLUSION

We designed a simple procedure for proving computational indistinguishability properties of protocols. In contrast with, e.g., the CIL [9], there is no need for probabilities: we rely simply on first-order logic. It remains to prove its usefulness in practice. There are however good clues that the inconsistency checks can be implemented efficiently. Indeed, the security property is a ground statement (no quantified variables), once the protocol is folded. Hence we only have to check the consistency of a (fixed) set of first-order formulas, together with a ground formula. It is likely that this can be performed in polynomial time, at least for a significant set of axioms (as it was the case for the more complicated situation of reachability properties [20]).

10. REFERENCES

[1] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *POPL'01*, pages 104–115. ACM, 2001.

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

[3] Martín Abadi and Cédric Fournet. Private authentication. *Theor. Comput. Sci.*, 322(3):427–476, September 2004.

[4] M. Backes, B. Pfitzmann, and M. Waidner. A composable cryptographic library with nested operations. In *CCS'03*, pages 220–230. ACM, 2003.

[5] G. Bana, P. Adão, and H. Sakurada. Computationally Comlete Symbolic Attacker in Action. In *FSTTCS'12*, LIPIcs, pages 546–560. Schloss Dagstuhl, 2012.

[6] G. Bana and H. Comon-Lundh. Towards unconditional soundness: Computationally complete symbolic attacker. In *POST'12*, LNCS, pages 189–208. Springer, 2012.

[7] G. Bana, K. Hasebe, and M. Okada. Computationally complete symbolic attacker and key exchange. In *CCS '13*, pages 1231–1246. ACM, 2013.

[8] G. Bana, P. Mohassel, and T. Stegers. Computational soundness of formal indistinguishability and static equivalence. In *ASIAN'06*, volume 4435 of *LNCS*, pages 182–196. Springer, 2007.

[9] G. Barthe, M. Daubignard, B. M. Kapron, and Y. Lakhnech. Computational indistinguishability logic. In *CCS'10*, pages 375–386. ACM, 2010.

[10] G. Barthe, B. Grégoire, S. Heraud, and S. Zanella-Béguelin. Computer-aided security proofs for the working cryptographer. In *CRYPTO 2011*, volume 6841 of *LNCS*, pages 71–90. Springer, 2011.

[11] M. Bellare, A. Boldyreva, A. Desai, and D. Pointcheval. Key-privacy in public-key encryption. In *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 566–582. Springer, 2001.

[12] M. Bellare, A. Desai, D. Pointcheval, and Ph. Rogaway. Relations among notions of security for public-key encryption schemes. In *CRYPTO'98*, LNCS. Springer, 1998.

[13] B. Blanchet. An automatic security protocol verifier based on resolution theorem proving (invited tutorial). In *CADE'05*, Tallinn, Estonia, July 2005.

[14] B. Blanchet. A computationally sound mechanized prover for security protocols. *IEEE Transactions on Dependable and Secure Computing*, 5(4):193–207, 2008.

[15] B. Blanchet, M. Abadi, and C. Fournet. Automated verification of selected equivalences for security protocols. *J. of Logic and Algebraic Programming*, 75(1):3–51, 2008.

[16] V. Cheval and B. Blanchet. Proving more observational equivalences with proverif. In *POST'13*, Lecture Notes in Computer Science, pages 226–246. Springer, 2013.

[17] Vincent Cheval, Hubert Comon-Lundh, and Stéphanie Delaune. Trace equivalence decision: Negative tests and non-determinism. In *CCS'11*, pages 321–330. ACM, 2011.

[18] H. Comon-Lundh and V. Cortier. Computational soundness of observational equivalence. In *CCS'08*, pages 109–118. ACM, 2008.

[19] H. Comon-Lundh, V. Cortier, and G. Scerri. Security proof with dishonest keys. In *POST'12*, LNCS, pages 149–168. Springer, 2012.

[20] H. Comon-Lundh, V. Cortier, and G. Scerri. Tractable inference systems: an extension with a deducibility predicate. In *CADE'13*, LNAI. Springer, 2013.

[21] Hubert Comon-Lundh, Masami Hagiya, Yusuke Kawamoto, and Hideki Sakurada. Computational soundness of indistinguishability properties without computable parsing. In *ISPEC'12*, pages 63–79. Springer, 2012.

[22] C. Cremers. The scyther tool: Verification, falsification, and analysis of security protocols. In *CAV'08*, volume 5123 of *LNCS*, pages 414–418. Springer, 2008.

[23] C. Ene, Y. Lakhnech, and V. C. Ng. Formal indistinguishability extended to the random oracle model. In *ESORICS'09*, volume 5789 of *LNCS*, pages 555–570. Springer, 2009.

[24] A. Armando et al. The AVISPA Tool for the automated validation of internet security protocols and applications. In *CAV'05*, volume 3576 of *LNCS*, pages 281–285, 2005.

[25] Bogdan Warinschi. A computational analysis of the needham-schroeder protocol. In *16th Computer security foundation workshop (CSFW)*, pages 248–262. IEEE, 2003.