Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science 186 (2007) 49–65
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

Explicit Randomness is not Necessary when Modeling Probabilistic Encryption [1]

Véronique Cortier Heinrich Hördegen Bogdan Warinschi

Loria/CNRS UMR 7503 & INRIA Lorraine projet Cassis & Université Henri-Poincaré & INPL Lorraine, France
{cortier, hordegen, warinsch}@loria.fr

Abstract

One of the most popular abstraction used in security analysis uses abstract, symbolic terms to model the bit strings sent over the network. However, the high level of abstraction blurs the significance of proofs carried out in such models with respect to real executions. In particular, although good encryption functions are randomized, most existing symbolic models for security do not capture explicitly the randomization of ciphertexts.
On the other hand, recent results relating symbolic models with cryptographic models require symbolic models where the randomization of ciphertexts is captured explicitly (through the use of labels attached to symbolic ciphertexts). Since little to no tool support exists for the resulting label-based models it may seem necessary to extend the decision procedures and the implementation of existing tools from the simpler models to the models that use labels.
In this paper we put forth a more practical alternative. We show that for a large class of security properties (that includes rather standard formulations of secrecy and authenticity), security of protocols with respect to the simpler model implies security in the model that uses labels. Combined with the computational soundness result of [4], our theorem enables the translation of security results obtained in symbolic models that do not use labels to standard computational security. Based on these results, we have recently implemented an AVISPA module for verifying security properties in a standard cryptographic model.

Keywords: Probabilistic encryption, security models, protocol verification, secrecy, authentication.

1 Introduction

Designers of mathematical models for computational systems need to find appropriate trade-offs between two seemingly contradictory requirements. Automatic verification (and thus usability) typically requires a high level of abstraction whereas prediction accuracy requires a high level of details. From this perspective, the use of symbolic models for security analysis is particularly delicate since it seems that the inherent high level of abstraction at which such models operate is not able to capture all aspects that are relevant to security. This paper is concerned with

[1] Work partially supported by the ACI Jeunes chercheurs and the ARA SSIA FormaCrypt.

doi:10.1016/j.entcs.2006.11.044

one particular such aspect, namely the use of randomization in the construction of cryptosystems [5].

A central feature of the computational, complexity-based models is the ability to capture and reason explicitly about the use of randomness. Moreover, randomness is essential to achieve any meaningful notion of security for encryption. In contrast, symbolic models rarely represent randomness directly. For example, a typical representation for the encryption of message m under the public key of entity B is the term $\{m\}_{\mathsf{ek}(B)}$. Notice that the symbolic representation does not capture the dependency on the randomness used to generate this ciphertext. While this abstraction may be sufficiently accurate in certain settings [11], in some other settings it is not sufficient.

Consider the following flow in some toy protocol:

$$A \to B : \{m\}_{\mathsf{ek}(B)}, \{\{m\}_{\mathsf{ek}(B)}\}_{\mathsf{ek}(B)}$$

To implement this flow, each occurrence of $\{m\}_{\mathsf{ek}(B)}$ is mapped to a ciphertext. Notice however that the pictorial description does not specify if the two occurrences of $\{m\}_{\mathsf{ek}(B)}$ are equal (created with identical randomness) or different (created with different randomness). In rich enough protocol specification languages disambiguating constructs as above can be easily done. For instance, in a language that has explicit assignments, the two different interpretation for the first message of the protocol can be obtained as

$$x := \{m\}_{\mathsf{ek}(B)}; \mathsf{send}(x, \{x\}_{\mathsf{ek}(B)}) \quad \text{and} \quad \mathsf{send}(\{m\}_{\mathsf{ek}(B)}, \{\{m\}_{\mathsf{ek}(B)}\}_{\mathsf{ek}(B)}).$$

Here, each distinct occurrence of $\{m\}_{\mathsf{ek}(B)}$ is interpreted with different randomness. Other approaches adopt a more direct solution and represent the randomness used for encryption explicitly [6, 1, 10, 4]. If we write $\{m\}^{l}_{\mathsf{ek}(B)}$ for the encryption of m under the public key of B with randomness l, the two different interpretations of the flow are:

$$\mathsf{send}(\{m\}^{l_1}_{\mathsf{ek}(B)}, \{\{m\}^{l_1}_{\mathsf{ek}(B)}\}^{l_2}_{\mathsf{ek}(B)}) \quad \text{and} \quad \mathsf{send}(\{m\}^{l_1}_{\mathsf{ek}(B)}, \{\{m\}^{l_2}_{\mathsf{ek}(B)}\}^{l_3}_{\mathsf{ek}(B)})$$

A model that employs labels to capture the randomness used in ciphertexts (and signatures) has recently been used to establish soundness of symbolic analysis with respect to computational models [4]. Their results are based on an emulation lemma: for protocol executions, every computational trace can be mapped to a valid symbolic trace. The mapping is then used to translate security properties that hold in the symbolic model to computational analogues. Note that the use of labels is necessary even when there is no explicit repetition of cyphertexts to distinguish for example the encrypted messages generated by the agents from those generated by the adversary.

The next step towards making the soundness result relevant to practice is to carry out the security proofs using some (semi-)automated tools for the symbolic model. However, to the best of our knowledge, none of the popular tools (ProVerif [3], CASPER [8], Athena [13], AVISPA [2]), offers capabilities for automatically reasoning in models that use labels. There are at least two solutions to this problem. One possibility is to enhance the symbolic models that underlie existing tools. Unfortunately such a modification would probably require significant effort that involves adapting existing decision procedures, proving their correctness, and verifying and modifying thousands of lines of code.

In this paper we put forth and clarify an alternative solution, used implicitly in [4]. The idea is to keep existing tools unchanged, use their underlying (unlabeled) model to prove security properties, and then show that the results are in fact meaningful for the model with labels. The main result of this paper is to prove that for a large class of security properties the approach that we propose is indeed feasible.

Results.

We consider the protocol specification language and the execution model developed in [4]. The language is for protocols that use random nonces, public key encryption and digital signatures, and uses labels to model the randomness used by these primitives. To each protocol Π with labels, we naturally associate a protocol $\overline{\Pi}$ obtained by erasing all labels, and extend the transformation to execution traces. To each trace tr of Π we associate a trace $\overline{tr}$ obtained by erasing labels and we extend this mapping to sets of traces. The first contribution of this paper is a proof that the transformation is sound. More precisely we prove that if tr is a valid trace of Π (obtained by Dolev-Yao operations) then $\overline{tr}$ is a valid trace of $\overline{\Pi}$. Importantly, this result relies on the fact that the specification language that we consider does not allow equality tests between ciphertexts. We believe that a similar result holds for most (if not all) protocol specification languages that satisfy the above condition. The language for specifying protocols (with and without labels) as well as the relation between their associated execution models are in Section 2.

In Section 3 we give two logics, $\mathcal{L}_1^l$ and $\mathcal{L}_1$, that we use to express security properties for protocols with and without labels, respectively. Informally, the formulas of $\mathcal{L}_1$ are obtained by removing the labels from formulas of $\mathcal{L}_1^l$. Both logics are quite expressive. For example, it can be used to express standard formulations for secrecy and authenticity properties.

Next we focus our attention on translating security properties between the two models. First, notice that the mapping between the model with and that without labels is not faithful since it looses information regarding inequality of ciphertexts. To formalize this intuition we give a protocol Π and a formula ϕ such that $\overline{\Pi}$ satisfies $\overline{\phi}$ (the formula that corresponds to ϕ in the model without labels), but for which Π does not satisfy ϕ. Anticipating, our example indicates that the source of problems is that ϕ may contain equality tests between ciphertexts, and such tests may not be translated faithfully. The counterexample is in Section 4.

The main result of the paper is a soundness theorem. We show that for a large class of security properties it is possible to carry out the proof in the model without labels and infer security properties in the model with labels. More precisely, we identify $\mathcal{L}_2^l$ and $\mathcal{L}_2$, fragments of $\mathcal{L}_1^l$ and $\mathcal{L}_1$ respectively, such that the following theorem holds.

Consider an arbitrary protocol Π and formula ϕ in $\mathcal{L}_2^l$. Let $\overline{\phi}$ be a formula in $\mathcal{L}_2$ obtained by erasing the labels that occur in ϕ. Then, it holds that:

$$\overline{\Pi} \models \overline{\phi} \implies \Pi \models \phi$$

The logics $\mathcal{L}_2^l$ and $\mathcal{L}_2$ are still expressive enough to contain the secrecy and authentication formulas. The theorem and its proof are in Section 4.

Based on our result, we implemented an AVISPA module [2] that is used to obtain computationally sound automatic proofs and used it to validate the protocols in the AVISPA library. The results of our experiments are described in Section 5.

2 Protocol

In this section we provide the syntax of protocols with labels. The presentation is adapted from [4]. The specification language is similar to the one of Casrul [12]; it allows parties to exchange messages built from identities and randomly generated nonces using public key encryption and digital signatures. Protocols that do not use labels are obtained straightforwardly.

2.1 Syntax

Consider an algebraic signature Σ with the following sorts. A sort ID for agent identities, sorts $\mathsf{SKey}, \mathsf{VKey}, \mathsf{EKey}, \mathsf{DKey}$ containing keys for signing, verifying, encryption, and decryption respectively. The algebraic signature also contains sorts Nonce, Label, $\mathsf{Ciphertext}$, $\mathsf{Signature}$ and Pair for nonces, labels, ciphertexts, signatures and pair, respectively. The sort Label is used in encryption and signatures to distinguish between different encryption/signature of the same plaintext. The sort Term is a supersort containing all other sorts, except SKey and DKey. There are nine operations: the four operations $\mathsf{ek}, \mathsf{dk}, \mathsf{sk}, \mathsf{vk}$ are defined on the sort ID and return the encryption key, decryption key, signing key, and verification key associated to the input identity. The two operations ag and adv are defined on natural numbers and return labels. As explained in the introduction, the labels are used to differentiate between different encryptions (and signatures) of the same plaintext, created by the honest agents or the adversary. We distinguish between labels for agents and for the adversary since they do not use the same randomness. The other operations that we consider are pairing, public key encryption, and signing.

We also consider sets of sorted variables $\mathsf{X} = \mathsf{X}.n \cup \mathsf{X}.a \cup \mathsf{X}.c \cup \mathsf{X}.s$ and $\mathsf{X}^l = \mathsf{X} \cup \mathsf{X}.l$. Here, $\mathsf{X}.n, \mathsf{X}.a, \mathsf{X}.c, \mathsf{X}.s, \mathsf{X}.l$ are sets of variables of sort nonce, agent, ciphertext, signature and labels, respectively. The sets of variables $\mathsf{X}.a$ and $\mathsf{X}.n$ are as follows. If $k \in \mathbb{N}$ is some fixed constant representing the number of protocol participants, w.l.o.g. we fix the set of agent variables to be $\mathsf{X}.a = \{A_1, A_2, \ldots, A_k\}$, and partition the set of nonce variables, by the party that generates them. Formally: $\mathsf{X}.n = \cup_{A \in \mathsf{X}.a} \mathsf{X}_n(A)$ and $\mathsf{X}_n(A) = \{X_A^j \mid j \in \mathbb{N}\}$. This partition avoids to specify later, for each role, which variables stand for generated nonces and which variables stand for expected nonces.

Labeled messages that are sent by participants are specified using terms in T^l

$$\begin{aligned}\mathcal{L} &::= X.l \mid \mathsf{ag}(i) \mid \mathsf{adv}(i)\\ T^l &::= \mathsf{X} \mid a \mid \mathsf{ek}(a) \mid \mathsf{dk}(a) \mid \mathsf{sk}(a) \mid \mathsf{vk}(a) \mid n(a,j,s)\\ &\qquad \mid \langle T^l \,, T^l \rangle \mid \{T^l\}^{\mathcal{L}}_{\mathsf{ek}(a)} \mid [T^l]^{\mathcal{L}}_{\mathsf{sk}(a)}\end{aligned}$$

where $i, j, s \in \mathbb{N}$ and $a \in \mathsf{ID}$.

Unlabeled messages are specified similarly as terms in the algebra T defined by

$$\begin{aligned}T ::= {}& \mathsf{X} \mid a \mid \mathsf{ek}(a) \mid \mathsf{dk}(a) \mid \mathsf{sk}(a) \mid \mathsf{vk}(a)\\ & \mid n(a,j,s) \mid \langle T \,, T \rangle \mid \{T\}_{\mathsf{ek}(a)} \mid [T]_{\mathsf{sk}(a)}\end{aligned}$$

where $j, s \in \mathbb{N}$ and $a \in \mathsf{ID}$.

A mapping $\bar{\cdot} : T^l \to T$ from labeled to unlabeled terms is defined by removing the labels: $\overline{\{k\}^l_m} = \{\overline{k}\}_{\overline{m}}$, $\overline{[k]^l_m} = [\overline{k}]_{\overline{m}}$, $\overline{f(t_1, \ldots, t_n)} = f(\overline{t_1}, \ldots, \overline{t_n})$ otherwise. The mapping function is extended to sets of terms as expected.

The individual behavior of each protocol participant is defined by a *role* that describes a sequence of message receptions/transmissions. A k-party protocol is given by k such roles.

Definition 2.1 [Labeled roles and protocols] The set Roles^l of roles for labeled protocol participants is defined by $\mathsf{Roles}^l = ((\{\mathsf{init}\} \cup T^l) \times (T^l \cup \{\mathsf{stop}\}))^*$. A k-party labeled protocol is a mapping $\Pi : [k] \to \mathsf{Roles}^l$, where $[k]$ denotes the set $\{1, 2, \ldots, k\}$.

Unlabeled roles and protocols are defined very similarly. The mapping function is extended from labeled protocols to unlabeled protocols as expected.

We assume that a protocol specification is such that $\Pi(j) = ((l^j_1, r^j_1), (l^j_2, r^j_2), \ldots)$, the j'th role in the definition of the protocol being executed by player A_j. Each sequence $((l_1, r_1), (l_2, r_2), \ldots) \in \mathsf{Roles}^l$ specifies the messages to be sent/received by the party executing the role: at step i, the party expects to receive a message conforming to l_i and returns message r_i. We wish to emphasize that terms l^j_i, r^j_i are not actual messages, but specify how the message that is received and the message that is output should look like.

Example 2.2 The Needham-Schroeder-Lowe protocol [7] is specified as follows: there are two roles $\Pi(1)$ and $\Pi(2)$ corresponding to the sender's and receiver's role.

$$\begin{aligned}A \to B &: \{N_a, A\}_{\mathsf{ek}(B)}\\ B \to A &: \{N_a, N_b, B\}_{\mathsf{ek}(A)}\\ A \to B &: \{N_b\}_{\mathsf{ek}(B)}\end{aligned}$$

$$\Pi(1) = (\mathsf{init}, \{X^1_{A_1}, A_1\}^{\mathsf{ag}(1)}_{\mathsf{ek}(A_2)}), (\{X^1_{A_1}, X^1_{A_2}, A_2\}^{L}_{\mathsf{ek}(A_1)}, \{X^1_{A_2}\}^{\mathsf{ag}(1)}_{\mathsf{ek}(A_2)})$$
$$\Pi(2) = (\{X^1_{A_1}, A_1\}^{L_1}_{\mathsf{ek}(A_2)}, \{X^1_{A_1}, X^1_{A_2}, A_2\}^{\mathsf{ag}(1)}_{\mathsf{ek}(A_1)}), (\{X^1_{A_2}\}^{L_2}_{\mathsf{ek}(A_2)}, \mathsf{stop})$$

Clearly, not all protocols written using the syntax above are meaningful. In particular, some protocols might be not executable. This is actually not relevant for our result (our theorem also holds for non executable protocols).

2.2 Execution Model

We define the execution model only for labeled protocols. The definition of the execution model for unlabeled protocols is then straightforward.

If A is a variable or constant of sort agent, we define its knowledge by $\mathbf{kn}(A) = \{\mathsf{dk}(A), \mathsf{sk}(A)\} \cup \mathsf{X}.n(A)$, *i.e.* an agent knows its secret decryption and signing key as well as the nonces it generates during the execution. The formal execution model is a state transition system. A *global state* of the system is given by (SId, f, H) where H is a set of terms of T^l representing the messages sent on the network and f maintains the local states of all session ids SId. We represent session ids as tuples of the form $(n, j, (a_1, a_2, \ldots, a_k)) \in (\mathbb{N} \times \mathbb{N} \times \mathsf{ID}^k)$, where $n \in \mathbb{N}$ identifies the session, $a_1, a_2, \ldots, a_k$ are the identities of the parties that are involved in the session and j is the index of the role that is executed in this session. Mathematically, f is a function $f : \mathsf{SId} \to ([\mathsf{X} \to T^l] \times \mathbb{N} \times \mathbb{N})$, where $f(\mathsf{sid}) = (\sigma, i, p)$ is the local state of session sid. The function σ is a partial instantiation of the variables occurring in role $\Pi(i)$ and $p \in \mathbb{N}$ is the control point of the program. Three transitions are allowed.

- $(\mathsf{SId}, f, H) \xrightarrow{\mathbf{corrupt}(a_1,\ldots,a_l)} (\mathsf{SId}, f, \cup_{1 \le j \le l} \mathbf{kn}(a_j) \cup H)$. The adversary corrupts parties by outputting a set of identities. He receives in return the secret keys corresponding to the identities. It happens only once at the beginning of the execution. We focus on static corruption because the soundness result using explicit labels in [4] only considers this kind of corruption. However, in our formal context, our reduction result should be easily extended to the case of adaptive corruption (when agents are corrupted at any time during the execution) since we can map traces with dynamic corruption to traces where all corrupted agents are so at the beginning.
- The adversary can initiate new sessions: $(\mathsf{SId}, f, H) \xrightarrow{\mathbf{new}(i,a_1,\ldots,a_k)} (\mathsf{SId}', f', H')$ where H', f' and SId' are defined as follows. Let $s = |\mathsf{SId}| + 1$, be the session identifier of the new session, where $|\mathsf{SId}|$ denotes the cardinality of SId. H' is defined by $H' = H$ and $\mathsf{SId}' = \mathsf{SId} \cup \{(s, i, (a_1, \ldots, a_k))\}$. The function f' is defined as follows.
  - $f'(\mathsf{sid}) = f(\mathsf{sid})$ for every $\mathsf{sid} \in \mathsf{SId}$.

$\dfrac{}{S \vdash^l m}\ m \in S$	$\dfrac{}{S \vdash^l b, \mathsf{ek}(b), \mathsf{vk}(b)}\ b \in \mathsf{X}.a$	Initial knowledge
$\dfrac{S \vdash^l m_1 \quad S \vdash^l m_2}{S \vdash^l \langle m_1\ , m_2\rangle}$	$\dfrac{S \vdash^l \langle m_1\ , m_2\rangle}{S \vdash^l m_i}\ i \in \{1,2\}$	Pairing and unpairing
$\dfrac{S \vdash^l \mathsf{ek}(b) \quad S \vdash^l m}{S \vdash^l \{m\}^{\mathsf{adv}(i)}_{\mathsf{ek}(b)}}\ i \in \mathbb{N}$	$\dfrac{S \vdash^l \{m\}^l_{\mathsf{ek}(b)} \quad S \vdash^l \mathsf{dk}(b)}{S \vdash^l m}$	Encryption and decryption
$\dfrac{S \vdash^l \mathsf{sk}(b) \quad S \vdash^l m}{S \vdash^l [m]^{\mathsf{adv}(i)}_{\mathsf{sk}(b)}}\ i \in \mathbb{N}$	$\dfrac{S \vdash^l [m]^l_{\mathsf{sk}(b)}}{S \vdash^l m}$	Signature

Fig. 1. Deduction rules.

· $f'(s, i, (a_1, \ldots, a_k)) = (\sigma, i, 1)$ where σ is a partial function $\sigma : \mathsf{X} \to T^l$ and:

$$\begin{cases} \sigma(A_j) & = a_j & 1 \leq j \leq k \\ \sigma(X^j_{A_i}) & = n(a_i, j, s) & j \in \mathbb{N} \end{cases}$$

We recall that the principal executing the role $\Pi(i)$ is represented by A_i thus, in that role, every variable of the form $X^j_{A_i}$ represents a nonce generated by A_i.

- The adversary can send messages: $(\mathsf{SId}, f, H) \xrightarrow{\mathbf{send}(\mathsf{sid},m)} (\mathsf{SId}, f', H')$ where $\mathsf{sid} \in \mathsf{SId}$, $m \in T^l$, H', and f' are defined as follows. We define $f'(\mathsf{sid}') = f(\mathsf{sid}')$ for every $\mathsf{sid}' \neq \mathsf{sid}$. We denote $\Pi(j) = ((l^j_1, r^j_1), \ldots, (l^j_{k_j}, r^j_{k_j}))$. $f(\mathsf{sid}) = (\sigma, j, p)$ for some σ, j, p. There are two cases.
  · Either there exists a most general unifier θ of m and $l^j_p\sigma$. Then $f'(\mathsf{sid}) = (\sigma \cup \theta, j, p+1)$ and $H' = H \cup \{r^j_p\sigma\theta\}$.
  · Or we define $f'(\mathsf{sid}) = f(\mathsf{sid})$ and $H' = H$ (the state remains unchanged).

If we denote by $\mathsf{SID} = \mathbb{N} \times \mathbb{N} \times \mathsf{ID}^k$ the set of all sessions ids, the set of *symbolic execution traces* is $\mathsf{SymbTr}^l = (\mathsf{SID} \times (\mathsf{SID} \to ([\mathsf{X} \to T^l] \times \mathbb{N} \times \mathbb{N})) \times 2^{T^l})^*$. The set of corresponding unlabeled symbolic execution traces is denoted by SymbTr. The mapping function $\bar{\cdot}$ is extended as follows: if $tr = (\mathsf{SId}_0, f_0, H_0), \ldots, (\mathsf{SId}_n, f_n, H_n)$ is a trace of SymbTr^l, $\overline{tr} = (\overline{\mathsf{SId}_0}, \overline{f_0}, \overline{H_0}), \ldots, (\overline{\mathsf{SId}_n}, \overline{f_n}, \overline{H_n}) \in \mathsf{SymbTr}$ where $\overline{\mathsf{SId}_i}$ simply equal SId_i and $\overline{f_i} : \mathsf{SID} \to ([\mathsf{X} \to T] \times \mathbb{N} \times \mathbb{N}))$ with $\overline{f_i}(\mathsf{sid}) = (\overline{\sigma}, i, p)$ if $f_i(\mathsf{sid}) = (\sigma, i, p)$ and $\overline{\sigma}(X) = \overline{\sigma(X)}$.

The adversary intercepts messages between honest participants and computes new messages using the deduction relation $\vdash^l$ defined in Figure 1. Intuitively, $S \vdash^l m$ means that the adversary is able to compute the message m from the set of messages S. All deduction rules are rather standard with the exception of the last one: The last rule states that the adversary can recover the corresponding message out of a given signature. This rule reflects capabilities that do not contradict the standard computational security definition of digital signatures, may potentially be available to computational adversaries and are important for the soundness result of [4].

Next, we sketch the execution model for unlabeled protocols. As above, the execution is based on a deduction relation $\vdash$ that captures adversarial capabilities. The deduction rules that define $\vdash$ are obtained from those of $\vdash^l$ (Figure 1) as follows. The sets of rules *Initial knowledge* and *Pairing and unpairing* in are kept unchanged (replacing $\vdash^l$ by $\vdash$, of course). For encryption and signatures we suppress the labels $\mathsf{adv}(i)$ and l in the encryption function $\{\_\}_{\_}^{\_}$ and the signature function $[\_]_{\_}^{\_}$ for rules *Encryption and decryption* and rules *Signature*. That is, the rules for encryption are:

$$\frac{S \vdash \mathsf{ek}(b) \quad S \vdash m}{S \vdash \{m\}_{\mathsf{ek}(b)}} \qquad \frac{S \vdash \{m\}_{\mathsf{ek}(b)} \quad S \vdash \mathsf{dk}(b)}{S \vdash m}$$

and those for signatures are:

$$\frac{S \vdash \mathsf{sk}(b) \quad S \vdash m}{S \vdash [m]_{\mathsf{sk}(b)}} \qquad \frac{S \vdash [m]_{\mathsf{sk}(b)}}{S \vdash m}$$

We use the deduction relations to characterize the set of valid execution traces. We say that the trace $(\mathsf{SId}_1, f_1, H_1), \ldots, (\mathsf{SId}_n, f_n, H_n)$ is *valid* if the messages sent by the adversary can be computed by Dolev-Yao operations. More precisely, we require that in a valid trace whenever $(\mathsf{SId}_i, f_i, H_i) \xrightarrow{\mathbf{send}(s,m)} (\mathsf{SId}_{i+1}, f_{i+1}, H_{i+1})$, we have $H_i \vdash^l m$. Given a protocol Π, the set of valid symbolic execution traces is denoted by $\mathsf{Exec}(\Pi)$. The set $\mathsf{Exec}(\overline{\Pi})$ of execution traces in the model without labels is defined similarly. We thus require that every sent message m' satisfies $\overline{H_i} \vdash m'$.

Example 2.3 Playing with the Needham-Schroeder-Lowe protocol described in Example 2.2, an adversary can corrupt an agent a_3, start a new session for the second role with players a_1, a_2 and send the message $\{n(a_3,1,1), a_1\}_{\mathsf{ek}(a_2)}^{\mathsf{adv}(1)}$ to the player of the second role. The corresponding valid trace execution is:

$$(\emptyset, f_1, \emptyset) \xrightarrow{\mathbf{corrupt}(a_3)} (\emptyset, f_1, \mathbf{kn}(a_3)) \xrightarrow{\mathbf{new}(2, a_1, a_2)}$$
$$(\{\mathsf{sid}_1\}, f_2, \mathbf{kn}(a_3)) \xrightarrow{\mathbf{send}(\mathsf{sid}_1, \{n_3, a_1\}_{\mathsf{ek}(a_2)}^{\mathsf{adv}(1)})}$$
$$\left(\{\mathsf{sid}_1\}, f_3, \mathbf{kn}(a_3) \cup \{\{n_3, n_2, a_2\}_{\mathsf{ek}(a_1)}^{\mathsf{ag}(1)}\}\right),$$

where $\mathsf{sid}_1 = (1, 2, (a_1, a_2))$, $n_2 = n(a_2, 1, 1)$, $n_3 = n(a_3, 1, 1)$, and f_2, f_3 are defined as follows: $f_2(\mathsf{sid}_1) = (\sigma_1, 2, 1)$, $f_3(\mathsf{sid}_1) = (\sigma_2, 2, 2)$ where $\sigma_1(A_1) = a_1$, $\sigma_1(A_2) = a_2$, $\sigma_1(X_{A_2}^1) = n_2$, and σ_2 extends σ_1 by $\sigma_2(X_{A_1}^1) = n_3$ and $\sigma_2(L_1) = \mathsf{adv}(1)$.

2.3 Relating the labeled and unlabeled execution models

The following lemma (which can be easily proved by structural induction) states that, whenever a message is deducible, the corresponding unlabeled message is also deducible.

Lemma 2.4 $S \vdash^l m \Rightarrow \overline{S} \vdash \overline{m}$

Based on the above property we show that whenever a trace corresponds to an execution of a protocol, the corresponding unlabeled trace corresponds also to an execution of the corresponding unlabeled protocol.

Lemma 2.5 $tr \in \mathsf{Exec}(\Pi) \Rightarrow \overline{tr} \in \mathsf{Exec}(\overline{\Pi})$.

Proof The key argument is that only pattern matching is performed in protocols and when a term with labels matches some pattern, the unlabeled term matches the corresponding unlabeled pattern. The proof is done by induction on the length of the trace. Full details are provided in Appendix A. $\square$

3 A logic for security properties

In this section we define a logic for specifying security properties. We then show that the logic is quite expressive and, in particular, it can be used to specify rather standard secrecy and authenticity properties.

3.1 Preliminary definitions

We define the set of local states $\mathcal{LS}_{i,p}(tr)$ of a trace tr for role i at step p by $\mathcal{LS}_{i,p}((\mathsf{SId}_k, f_k, H_k)_{1\le k\le n}) = \{(\sigma, i, p) \mid \exists s \in \mathsf{SId}_k, \text{ s.t. } f_k(s) = (\sigma, i, p), 1 \le k \le n\}$.

We assume an infinite set *Sub* of meta-variables for substitutions. Our logic contains tests between terms with variables substituted by variable substitutions. More formally, let T^l_{Sub} be the algebra defined by:

$$\begin{aligned} \mathcal{L} \quad &::= \varsigma(x_l) \mid \mathsf{ag}(i) \mid \mathsf{adv}(j) \\ T^l_{Sub} &::= \varsigma(x) \mid a \mid \mathsf{ek}(a) \mid \mathsf{dk}(a) \mid \mathsf{sk}(a) \mid \mathsf{vk}(a) \mid n(a,j,s) \\ &\qquad \mid \langle T^l_{Sub}, T^l_{Sub}\rangle \mid \{T^l_{Sub}\}^{\mathcal{L}}_{\mathsf{ek}(a)} \mid [T^l_{Sub}]^{\mathcal{L}}_{\mathsf{sk}(a)} \end{aligned}$$

where $x_l \in \mathsf{X}.l$, $\varsigma \in Sub$, $i, j \in \mathbb{N}$, $x \in \mathsf{X}$, $a \in \mathsf{ID}$. The unlabeled algebra T_{Sub} is defined accordingly. The mapping function between the two algebras is defined by: $\overline{\varsigma(x)} = \varsigma(x)$, $\overline{\{k\}^l_m} = \{\overline{k}\}_{\overline{m}}$, $\overline{[k]^l_m} = [\overline{k}]_{\overline{m}}$, $\overline{f(t_1,\ldots,t_n)} = f(\overline{t_1},\ldots,\overline{t_n})$ otherwise.

3.2 Security Logic

In this section we describe a logic for security properties. Besides standard propositional connectors, the logic has a predicate to specify honest agents, equality tests between terms, and existential and universal quantifiers over the local states of agents.

$$[\![NC(tr,t)]\!] = \begin{cases} 1 \text{ if } t \in \mathsf{ID} \text{ and } t \text{ does not appear in a corrupt action, } i.e.\ tr = e_1, e_2, ..., e_n \text{ and } \\ \quad \forall a_1, \ldots, a_k, \text{ s.t. } e_1 \xrightarrow{\textbf{corrupt}(a_1,...,a_k)} e_2, t \neq a_i, \\ 0 \text{ otherwise} \end{cases}$$

$$[\![\forall \mathcal{LS}_{i,p}(tr).\varsigma\ F(tr)]\!] = \begin{cases} 1 \text{ if } \forall(\theta, i, p) \in \mathcal{LS}_{i,p}(tr), \text{ we have } \\ \quad [\![F(tr)[\theta/\varsigma]]\!] = 1, \\ 0 \text{ otherwise.} \end{cases}$$

$$[\![\exists \mathcal{LS}_{i,p}(tr).\varsigma\ F(tr)]\!] = \begin{cases} 1 \text{ if } \exists(\theta, i, p) \in \mathcal{LS}_{i,p}(tr), \text{ s.t. } [\![F(tr)[\theta/\varsigma]]\!] = 1, \\ 0 \text{ otherwise.} \end{cases}$$

Fig. 2. Interpretation.

Definition 3.1 The formulas of the logic $\mathcal{L}_1^l$ are defined by induction as follows:

$$\begin{aligned} F(tr) ::= &\ NC(tr, t_1) \mid (t_1 = t_2) \mid \neg F(tr) \mid F(tr) \wedge F(tr) \mid F(tr) \vee F(tr) \\ &\mid \forall \mathcal{LS}_{i,p}(tr).\varsigma\ F(tr) \mid \exists \mathcal{LS}_{i,p}(tr).\varsigma\ F(tr) \end{aligned}$$

where tr is a parameter of the formula, $i, p \in \mathbb{N}$, $\varsigma \in Sub$, t_1 and t_2 are terms of T_{Sub}^l. Note that formulas are parametrized by a trace tr. As usual, we may use $\phi_1 \rightarrow \phi_2$ as a shortcut for $\neg\phi_1 \vee \phi_2$.

We similarly define the corresponding unlabeled logic $\mathcal{L}_1$: the tests $(t_1 = t_2)$ are between unlabeled terms t_1, t_2 over T_{sub}. The mapping function $\bar{\cdot}$ is extended as expected. In particular $\overline{NC(tr,t)} = NC(\overline{tr}, \overline{t})$, $\overline{(t_1 = t_2)} = (\overline{t_1} = \overline{t_2})$, $\overline{\forall \mathcal{LS}_{i,p}(tr).\varsigma\ F(tr)} = \forall \mathcal{LS}_{i,p}(\overline{tr}).\varsigma\ \overline{F(tr)}$ and $\overline{\exists \mathcal{LS}_{i,p}(tr).\varsigma\ F(tr)} = \exists \mathcal{LS}_{i,p}(\overline{tr}).\varsigma\ \overline{F(tr)}$.

Here the predicate $NC(tr, t)$ of arity 2 is used to specify non corrupted agents. The quantifications $\forall \mathcal{LS}_{i,p}(tr).\varsigma$ and $\exists \mathcal{LS}_{i,p}(tr).\varsigma$ are over local states of agent i at step p in trace tr. The semantics of our logic is defined for *closed* formula as follows: standard propositional connectors and negation are interpreted as usual. Equality is syntactic equality. The interpretation of quantifiers and the predicate NC is shown in Figure 2.

Next we define when a protocol Π satisfies a formula $\phi \in \mathcal{L}_1^l$. The definition for the unlabeled execution model is obtained straightforwardly. Informally, a protocol Π satisfies $\phi(tr)$ if $\phi(tr)$ is true for all traces of Π. Formally:

Definition 3.2 Let $\phi(tr)$ be a formula and Π be a protocol. We say that Π satisfies security property ϕ, and write $\Pi \models \phi$ if for any trace $tr \in \mathsf{Exec}(\Pi)$, $[\![\phi(tr)]\!] = 1$.

Abusing notation, we occasionally write ϕ for the set $\{tr \mid [\![\phi(tr)]\!] = 1\}$. Then, $\Pi \models \phi$ precisely when $\mathsf{Exec}(\Pi) \subseteq \phi$.

3.3 Examples of security properties

In this section we exemplify the use of the logic by specifying secrecy and authenticity properties.

3.3.1 A secrecy property

Let $\Pi(1)$ and $\Pi(2)$ be the sender's and receiver's role of a two-party protocol. To specify our secrecy property we use a standard encoding. Namely, we add a third role to the protocol, $\Pi(3) = (X^1_{A_3}, \mathit{stop})$, which can be seen as some sort of witness.

Informally, the definition of the secrecy property P_s states that, for two non corrupted agents a_1 and a_2, where a_1 plays role $\Pi(1)$ and a_2 plays role $\Pi(2)$, a third agent playing role $\Pi(3)$ cannot gain any knowledge on nonce $X^1_{A_1}$ sent by role $\Pi(1)$ (played by A_1), when A_1 is honest and is talking with an honest agent A_2.

$$\phi_s(tr) = \forall \mathcal{LS}_{1,1}(tr).\varsigma\ \forall \mathcal{LS}_{3,2}(tr).\varsigma' \\ [NC(tr, \varsigma(A_1)) \wedge NC(tr, \varsigma(A_2)) \rightarrow \neg(\varsigma'(X^1_{A_3}) = \varsigma(X^1_{A_2}))]$$

3.3.2 An authentication property

Consider a two role protocol, such that role 1 finishes its execution after n steps and role 2 finishes its execution after p steps. For this kind of protocols we give a variant of the week agreement property [9]. Informally, this property states that whenever an instantiation of role 2 finishes, there exists an instantiation of role 1 that has finished and they agree on some value for some variable and they have indeed talked to each other. In our example we choose this variable to be $X^1_{A_1}$. Note that we capture that some agent has finished its execution by quantifying appropriately over the local states of that agent. More precisely, we quantify only over the states where it indeed has finished its execution.

$$\phi_a(tr) = \forall \mathcal{LS}_{2,p}(tr).\varsigma\ \exists \mathcal{LS}_{1,n}(tr).\varsigma'\ [NC(tr, \varsigma(A_1)) \wedge NC(tr, \varsigma'(A_2)) \rightarrow \\ (\varsigma(X^1_{A_1}) = \varsigma'(X^1_{A_1})) \wedge (\varsigma(A_2) = \varsigma'(A_2)) \wedge (\varsigma(A_1) = \varsigma'(A_1))]$$

Notice that although in its current version our logic is not powerful enough to specify stronger versions of agreement (like injective or bijective agreement), it could be appropriately extended to deal with this more complex forms of authentication.

4 Main Result

Recall that our goal is to prove that $\overline{\Pi} \models \overline{\phi} \Rightarrow \Pi \models \phi$. However, as explained in the introduction this property does not hold in general. The following example sheds some light on the reasons that cause the desired implication to fail.

Example 4.1 Consider the first step of some protocol where A sends a message to B where some part is intended for some third agent.

$$A \rightarrow B : \{N_a, \{N_a\}_{\mathsf{ek}(C)}, \{N_a\}_{\mathsf{ek}(C)}\}_{\mathsf{ek}(B)}$$

The specification of the roles of A and B that corresponds to this first step is as follows (in the definition below $C^1_{A_2}$ and $C^2_{A_2}$ are variables of sort ciphertext).

$$\begin{aligned}\Pi(1) &= (\mathsf{init}, \{\langle X^1_{A_1}, \langle \{X^1_{A_1}\}^{\mathsf{ag}(1)}_{\mathsf{ek}(A_3)}, \{X^1_{A_1}\}^{\mathsf{ag}(2)}_{\mathsf{ek}(A_3)}\rangle\rangle\}^{\mathsf{ag}(3)}_{\mathsf{ek}(A_2)})\\ \Pi(2) &= (\{\langle X^1_{A_1}, \langle C^1_{A_2}, C^2_{A_2}\rangle\rangle\}^{L}_{\mathsf{ek}(A_2)}, \mathsf{stop})\end{aligned}$$

We assume that A generates twice the message $\{N_a\}_{\mathsf{ek}(C)}$. Notice that we stop the execution of B after it receives the first message since this is sufficient for our purpose, but its execution might be continued to form a more realistic example.

Consider the security property ϕ_1 that states that if A and B agree on the nonce $X^1_{A_1}$ then B should have received twice the same ciphertext.

$$\begin{aligned}\phi_1(tr) = \forall \mathcal{LS}_{1,2}(tr).\varsigma\ \forall \mathcal{LS}_{2,2}(tr).\varsigma'&\\ NC(tr, \varsigma(A_1)) \wedge NC(tr, \varsigma(A_2)) \wedge&\\ (\varsigma(X^1_{A_1}) = \varsigma'(X^1_{A_1})) \to (\varsigma'(C^1_{A_2}) = \varsigma'(C^2_{A_2}))&\end{aligned}$$

This property clearly does not hold for any normal execution of the labeled protocol since A always sends ciphertexts with distinct labels. Thus $\Pi \not\models \phi_1$.

On the other hand, one can show that we have $\overline{\Pi} \models \overline{\phi_1}$ in the unlabeled execution model. Intuitively, this holds because if A and B are honest agents and agree on $X^1_{A_1}$, then the message received by B has been emitted by A and thus should contain identical ciphertexts (after having removed their labels).

4.1 Logic $\mathcal{L}^l_2$

The counterexample above relies on the fact that two ciphertexts that are equal in the model without labels may have been derived from distinct ciphertexts in the model with labels. Hence, it may be the case that although $\overline{t_1} \neq \overline{t_2} \Rightarrow t_1 \neq t_2$, the contra-positive implication $\overline{t_1} = \overline{t_2} \Rightarrow t_1 = t_2$ does not hold, which in turn entails that formulas that contain equality tests between ciphertexts may be true in the model without labels, but false in the model with labels. In this section we identify a fragment of $\mathcal{L}^l_1$, which we call $\mathcal{L}^l_2$ where such tests are prohibited. Formally, we avoid equality tests between arbitrary terms by forbidding arbitrary negation over formulas and allowing equality tests only between *simple* terms.

Definition 4.2 A term t is said *simple* if $t = \varsigma(x)$ where $x \in \mathsf{X}.a \cup \mathsf{X}.n$ and $\varsigma \in Sub$, or $t = a$ for some $a \in \mathsf{ID}$ or $t = n(a, j, s)$ for some $a \in \mathsf{ID}$, $j, s \in \mathbb{N}$.

An important observation is that for any simple term t it holds that $\overline{t} = t$.

Definition 4.3 The formulas of the logic $\mathcal{L}^l_2$ are defined as follows:

$$\begin{aligned}F(tr) ::= NC(tr, t_1) \mid \neg NC(tr, t_1) \mid F(tr) \wedge F(tr) \mid F(tr) \vee F(tr) \mid (t_1 \neq t_2)\\ \mid (u_1 = u_2) \mid \forall \mathcal{LS}_{i,p}(tr).\varsigma\ F(tr) \mid \exists \mathcal{LS}_{i,p}(tr).\varsigma\ F(tr),\end{aligned}$$

where $tr \in \mathsf{SymbTr}$ is a parameter, $i, p \in \mathbb{N}$, $t_1, t_2 \in T^l_{Sub}$ and u_1, u_2 are simple terms.

Since simple terms also belong to T^l_{Sub}, both equality and inequality tests are allowed between simple terms.

The corresponding unlabeled logic $\mathcal{L}_2$ is defined as expected. Note that $\mathcal{L}^l_2 \subset \mathcal{L}^l_1$ and $\mathcal{L}_2 \subset \mathcal{L}_1$.

4.2 Theorem

Informally, our main theorem says that to verify if a protocol satisfies some security formula ϕ in logic $\mathcal{L}^l_2$, it is sufficient to verify that the unlabeled version of the protocol satisfies $\overline{\phi}$.

Theorem 4.4 *Let Π be a protocol and $\phi \in \mathcal{L}^l_2$, then $\overline{\Pi} \models \overline{\phi} \Rightarrow \Pi \models \phi$.*

Proof Assume $\overline{\Pi} \models \overline{\phi}$. We have to show that for any trace $tr \in \mathsf{Exec}(\Pi)$, $[\![\phi(tr)]\!] = 1$. From Lemma 2.5 it follows that $\overline{tr} \in \mathsf{Exec}(\overline{\Pi})$, thus $[\![\overline{\phi}(\overline{tr})]\!] = 1$, since $\overline{\Pi} \models \overline{\phi}$. Thus, it is sufficient to show that $[\![\overline{\phi}(\overline{tr})]\!] = 1 \Rightarrow [\![\phi(tr)]\!] = 1$. The following lemma offers the desired property. □

Lemma 4.5 *Let $\phi(tr) \in \mathcal{L}^l_2$ for some $tr \in \mathsf{SymbTr}$, $[\![\overline{\phi}(\overline{tr})]\!] = 1$ implies $[\![\phi(tr)]\!] = 1$.*

Proof The proof of the lemma is by induction on the structure of $\phi(tr)$. Full details are provided in Appendix B. □

5 Implementation and Experiments

The AVISPA project [2] provides a platform for automatic verification of security protocols. The platform includes a specification language called HLPSL that can be used for specifying both protocols and security properties. Protocols specified in this language can be verified with four different tools. Three of them, OFMC, ATSE, and SATMC, use symbolic models where the number of sessions that can be executed in parallel is bounded. The fourth tool, TA4SP, provides verification abilities for an unbounded number of sessions. The tools can be used to verify three security properties: secrecy, weak authentication, and replay protection.

Based on the results of [4] and this paper, we implemented a module for the AVISPA with the purpose of obtaining computationally sound security guarantees. The module works as follows. First, the module verifies that the protocol (specified in HLPSL) can be translated in our formalism. In particular, it verifies that the protocol uses only asymmetric encryption in its pure form (that is: we do not consider protocols that model digital signatures via decryption-with-the-private key approach.) Notice that although we forbid protocols that encrypt messages using symmetric keys, such keys can still be sent around. Next, the module checks whether the security property that is verified can be translated in our $\mathcal{L}_2$ logic. In particular, weak authentication should only be done on atomic messages like agents and

Public keys only	symbolically secure	computationally secure
13	9	9

Fig. 3. Summary of our experiments.

nonces. Finally, if the verification succeeds the logical formula that states the security property is printed out, together with a messages that states that the protocol satisfies the security property computationally.

We executed the module on the protocols in the library of the AVISPA platform. The results are summarized in Figure 3. Of the 13 public-key encryption based protocols in the library of the AVISPA platform the tool concludes that 9 are symbolically secure, and all 9 pass our syntactic validation tests. We conclude that all these 9 protocols are computationally secure.

The new module will be included in the next version of the AVISPA tool.

6 Discussion

We conclude with a brief discussion of two interesting aspects of our result. First, as mentioned in the introduction our main theorem should hold for all execution models for which the underlying deduction systems satisfy the condition in Lemma 2.4, that is $S \vdash^l m \Rightarrow \overline{S} \vdash \overline{m}$. For example, it should hold for the deduction systems obtained after removing the rule

$$\frac{S \vdash^l [m]^l_{\mathsf{sk}(b)}}{S \vdash^l m}$$

and its corresponding unlabeled variant. In fact, an interesting result would be to prove a more abstract and modular version of our theorem.

Secondly, one may ask if the converse of our main theorem holds. We argue that this is not the case. More precisely, we show that there exists a protocol Π and a property ϕ such that $\Pi \models \phi$ but $\overline{\Pi} \not\models \overline{\phi}$. Let Π be the protocol defined in Example 4.1. Consider a security property ϕ_2 that states on the contrary that whenever A and B agree on the nonce $X^1_{A_1}$ then B should have received two distinct ciphertexts. Formally:

$$\begin{aligned}\phi_2(tr) = \forall \mathcal{LS}_{1,2}(tr).\varsigma\ \forall \mathcal{LS}_{2,2}(tr).\varsigma' \\ NC(tr, \varsigma(A_1)) \wedge NC(tr, \varsigma(A_2)) \wedge \\ (\varsigma(X^1_{A_1}) = \varsigma'(X^1_{A_1})) \rightarrow (\varsigma'(C^1_{A_2}) \neq \varsigma'(C^2_{A_2}))\end{aligned}$$

where $C^1_{A_2}$ and $C^2_{A_2}$ are variables of sort ciphertext.

This property clearly does not hold for any honest execution of the unlabeled protocol since A always sends twice the same ciphertext, and thus $\overline{\Pi} \not\models \overline{\phi_2}$. On the other hand however, one can show that this property holds for labeled protocols since, if A and B are honest agents and agree on $X^1_{A_1}$, it means that the message received by B has been emitted by A and thus contains two distinct ciphertexts. Thus, $\Pi \models \phi_2$. We conclude that, in general, $\Pi \models \phi$ does not imply $\overline{\Pi} \models \overline{\phi}$.

References

[1] Abadi, M. and Jürjens, J. (2001). Formal eavesdropping and its computational interpretation. In *Proc. of Theoretical Aspects of Computer Software (TACS 2001)*, volume 2215 of *LNCS*, pages 82–94. Springer-Verlag.

[2] Armando, A., Basin, D., Boichut, Y., Chevalier, Y., Compagna, L., Cuellar, J., Hankes Drielsma, P., Héam, P.-C., Kouchnarenko, O., Mantovani, J., Mödersheim, S., von Oheimb, D., Rusinowitch, M., Santiago, J., Turuani, M., Viganò, L., and Vigneron, L. (2005). The AVISPA Tool for the automated validation of internet security protocols and applications. In *17th International Conference on Computer Aided Verification, CAV'2005*, volume 3576 of *LNCS*, pages 281–285. Springer.

[3] Blanchet, B. (2001). An efficient cryptographic protocol verifier based on prolog rules. In *Proc. of the 14th Computer Security Foundations Workshop (CSFW'01)*.

[4] Cortier, V. and Warinschi, B. (2005). Computationally Sound, Automated Proofs for Security Protocols. In *Proc. 14th European Symposium on Programming (ESOP'05)*, volume 3444 of *Lecture Notes in Computer Science*, pages 157–171. Springer.

[5] Goldwasser, S. and Micali, S. (1984). Probabilistic encryption. *J. of Computer and System Sciences*, 28:270–299.

[6] Herzog, J. C. (2004). *Computational Soundness for Standard Asumptions of Formal Cryptography.* PhD thesis, Massachusetts Institute of Technology.

[7] Lowe, G. (1996). Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proc. of Tools and algoritms for the construction and analysis fof systems (TACAS'96)*, volume 1055 of *LNCS*, pages 147–166. Springer-Verlag.

[8] Lowe, G. (1997a). Casper: A compiler for the analysis of security protocols. In *Proc. of 10th Computer Security Foundations Workshop (CSFW'97)*. IEEE Computer Society Press.

[9] Lowe, G. (1997b). A hierarchy of authentication specifications. In *Proc. of the 10th Computer Security Foundations Workshop (CSFW'97)*. IEEE Computer Society Press.

[10] Lowe, G. (2004). Analysing protocols subject to guessing attacks. *Journal of Computer Security*, 12(1).

[11] Micciancio, D. and Warinschi, B. (2004). Soundness of formal encryption in the presence of active adversaries. In *Theory of Cryptography Conference (TCC 2004)*, pages 133–151. Springer-Verlag.

[12] Rusinowitch, M. and Turuani, M. (2001). Protocol insecurity with finite number of sessions is NP-complete. In *Proc. of the 14th Computer Security Foundations Workshop (CSFW'01)*, pages 174–190. IEEE Computer Society Press.

[13] Song, D. X. (1999). Athena: A new efficient automatic checker for security protocol analysis. In *Proc. of the 12th Computer Security Foundations Workshop (CSFW'99)*. IEEE Computer Society Press.

A Proof of Lemma 2.5

Lemma 2.5 $tr \in \mathsf{Exec}(\Pi) \Rightarrow \overline{tr} \in \mathsf{Exec}(\overline{\Pi})$.

Proof The key argument is that only pattern matching is performed in protocols and when a term with labels matches some pattern, the unlabeled term matches the corresponding unlabeled pattern.

- Let $tr = (\mathsf{SId}_0, f_0, H_0)$, where SId_0 and H_0 are empty sets. We have $\overline{H_0} = H_0$. f_0 is defined nowhere, and so is $\overline{f_0}$. Clearly, $\overline{tr} = (\mathsf{SId}_0, \overline{f_0}, \overline{H_0})$ is in $\mathsf{Exec}(\overline{\Pi})$.
- Let $tr \in \mathsf{Exec}(\Pi)$, $tr = e_0, ..., e_n = (\mathsf{SId}_0, f_0, H_0), ..., (\mathsf{SId}_n, f_n, H_n)$, such that $\overline{tr} \in \mathsf{Exec}(\overline{\Pi})$. We have to show that if $tr' = tr, (\mathsf{SId}_{n+1}, f_{n+1}, H_{n+1}) \in \mathsf{Exec}(\Pi)$, then we have $\overline{tr'} \in \mathsf{Exec}(\overline{\Pi})$. There are three possible operations.

(i) $corrupt(a_1, ..., a_k)$. It means that $tr = (\mathsf{SId}_0, f_0, H_0), (\mathsf{SId}_1, f_1, H_1)$. In this case, we have $\mathsf{SId}_1 = \mathsf{SId}_0 = \emptyset$, $f_1 = f_0$ and $H_1 = H_0 \cup \bigcup_{1 \leq i \leq k} \mathbf{kn}(a_i)$. We can conclude that $\overline{tr} = (\mathsf{SId}_0, \overline{f_0}, \overline{H_0}), (\mathsf{SId}_1, \overline{f_1}, \overline{H_1})$ is in $\mathsf{Exec}(\overline{\Pi})$, because there are no labels in H_1 and f_1 is still not defined.

(ii) $new(i, a_1, ..., a_k)$. No labels are involved in this operation. The extension made to f_n is the same as is made to $\overline{f_n}$. Neither H_n nor $\overline{H_n}$ are modified. $\overline{tr'} = \overline{tr}, (\mathsf{SId}_{n+1}, \overline{f_{n+1}}, \overline{H_{n+1}})$ is a valid trace.

(iii) $send(s, m)$.

First, we have to be sure that if m can be deduced from H_n, then $\overline{m}$ can be deduced from $\overline{H_n}$. This is Lemma 2.4.

Note that $\mathsf{SId}_n = \mathsf{SId}_{n+1}$ thus $\overline{\mathsf{SId}_n} = \overline{\mathsf{SId}_{n+1}}$. Let $f_n(s) = (\sigma, i, p)$ and $\Pi(i) = (..., (l_p, r_p), ...)$. We have two cases.

- Either there is a substitution θ with $m = l_p\sigma\theta$. Then $f_{n+1}(s) = (\sigma \cup \theta, i, p+1)$. Thus $\overline{f_n}(s) = (\overline{\sigma}, i, p)$ and $\overline{f_{n+1}}(s) = (\overline{\sigma} \cup \overline{\theta}, i, p+1)$. By induction hypothesis, $\overline{tr}$ is a valid trace. From $m = l_p\sigma\theta$ follows $\overline{m} = \overline{l_p}\overline{\sigma}\overline{\theta}$. We conclude that $\overline{tr}, (\mathsf{SId}_{n+1}, \overline{f_{n+1}}, \overline{H_{n+1}}) = \overline{tr'}$ is a valid trace, thus a member of $\mathsf{Exec}(\overline{\Pi})$.
- Or no substitution θ with $m = l_p\sigma\theta$ exists. Then $tr' = e_0, ..., e_n, e_{n+1}$ with $e_n = e_{n+1}$. We must show that it is always possible to construct a message $m' \in T$, such that there exists no substitution θ' with $m' = \overline{l_p}\overline{\sigma}\theta'$. Then, from the validity of tr' and $\overline{tr}$ we can deduce the validity of $\overline{tr'}$, because $\overline{e_n} = \overline{e_{n+1}}$.

  Either there exists no substitution θ' such that $\overline{m} = \overline{l_p}\overline{\sigma}\theta'$. In that case, we choose $m' = \overline{m}$.

  Or let θ' be a substitution such that $\overline{m} = \overline{l_p}\overline{\sigma}\theta'$. Then the matching for m fails because of labels. This can be shown by contradiction. Assume m contain no label, *i. e.* m does not contain subterms of the form $\{t\}^l_{\mathsf{ek}(a_i)}$ or $[t]^l_{\mathsf{sk}(a_i)}$, $t \in T$. In that case, we have $\overline{m} = m$ by definition. From $\overline{m} = \overline{l_p}\overline{\sigma}\theta'$, we deduce that $m = l_p\sigma\theta'$, contradiction.

  We deduce that $\overline{m}$ contains some subterm of the form $\{t\}_{\mathsf{ek}(a_i)}$ or $[t]_{\mathsf{sk}(a_i)}$. The fact $\overline{m} = \overline{l_p}\overline{\sigma}\theta'$ implies that $\overline{l_p}$ has to contain one of the following subterms: $\{t'\}_{\mathsf{ek}(A_i)}$, $[t']_{\mathsf{sk}(A_i)}$ with $t' \in T$ or, a variable of sort ciphertext or signature.

  Then, we choose $m' = a$ for some agent identity $a \in \mathsf{X}.a$. The term a is deducible from $\overline{H_n}$. Now, the matching of m' with $\overline{l_p}$ always fails, either because of the encryption or signature occurring in $\overline{l_p}$ or because of type mismatch for a variable of type ciphertext or signature in $\overline{l_p}$. □

B Proof of Lemma 4.5

Lemma 4.5 *Let $\phi(tr) \in \mathcal{L}^l_2$ for some $tr \in \mathsf{SymbTr}$, $[\![\overline{\phi}(\overline{tr})]\!]$ implies $[\![\phi(tr)]\!]$.*

Proof

- $\phi(tr) = NC(tr, t)$ or $\phi(tr) = \neg NC(tr, t)$. $[\![NC(tr, t)]\!] = 1$, if and only if $t \in \mathsf{ID}$ and t does not occur in a **corrupt** event for the trace tr. This is equivalent to $\overline{t} \in \mathsf{ID}$ and $\overline{t}$ does not occur in a **corrupt** event for the trace $\overline{tr}$. Thus $[\![NC(tr, t)]\!] = 1$ if and only if $[\![\overline{NC(tr, t)}]\!] = [\![NC(\overline{tr}, \overline{t})]\!] = 1$.
- $\phi(tr) = (t_1 \neq t_2)$. We have that $\overline{\phi}(\overline{tr}) = (\overline{t_1} \neq \overline{t_2})$ holds. Assume by contradiction that $\phi(tr)$ does not hold, *i.e* $t_1 = t_2$. This implies $\overline{t_1} = \overline{t_2}$, contradiction.
- $\phi(tr) = (u_1 = u_2)$ with u_1, u_2 simple terms. We have that $\overline{\phi}(\overline{tr}) = (\overline{u_1} = \overline{u_2})$

holds. Since u_1 and u_2 are simple terms, we have $\overline{u_i} = u_i$, thus $u_1 = u_2$. We conclude that $\phi(tr)$ holds.

- The cases $\phi(tr) = \phi_1(tr) \vee \phi_2(tr)$ and $\phi(tr) = \phi_1(tr) \wedge \phi_2(tr)$ are straightforward.
- $\phi(tr) = \forall \mathcal{LS}_{i,p}(tr).\varsigma\ F(tr)$. If $\overline{\phi}(\overline{tr})$ holds, this means that for all $(\theta, i, p) \in \mathcal{LS}_{i,p}(\overline{tr})$, $[\![\overline{F}(\overline{tr})[\theta/\varsigma]]\!] = 1$.

  Let $(\theta', i, p) \in \mathcal{LS}_{i,p}(tr)$. We consider $[\![F(tr)[\theta'/\varsigma]]\!]$. Since $tr \in \mathsf{Exec}(\Pi)$ implies $\overline{tr} \in \mathsf{Exec}(\overline{\Pi})$ (Lemma 2.5), we have $(\overline{\theta'}, i, p) \in \mathcal{LS}_{i,p}(\overline{tr})$. By induction hypothesis, $[\![\overline{F}(\overline{tr})[\overline{\theta'}/\varsigma]]\!] = 1$ implies that $[\![F(tr)[\theta'/\varsigma]]\!] = 1$. It follows that

$$\forall(\theta', i, p) \in \mathcal{LS}_{i,p}(tr)\ [\![F(tr)[\theta'/\varsigma]]\!] = 1.$$

  Thus, $\phi(tr)$ holds.
- $\phi(tr) = \exists \mathcal{LS}_{i,p}(tr).\varsigma\ F(tr)$. If $\overline{\phi}(\overline{tr})$ holds, this means that there exists $(\theta, i, p) \in \mathcal{LS}_{i,p}(\overline{tr})$, such that $[\![\overline{F}(\overline{tr})[\theta/\varsigma]]\!] = 1$.

  By definition of the mapping function, there exists $(\theta', i, p) \in \mathcal{LS}_{i,p}(tr)$ such that $\overline{\theta'} = \theta$. By induction hypothesis, $[\![F(tr)[\theta'/\varsigma]]\!] = 1$. Thus there exists θ', such that $[\![F(tr)[\theta'/\varsigma]]\!] = 1$. Thus, $\phi(tr)$ holds. $\square$