Bounding messages for free in security protocols - Extension to various security properties

Myrto Arapinis[a], Marie Duflot[b,*]

[a] *University of Edinburgh, UK*
[b] *LORIA & Université de Lorraine, France*

Abstract

While the verification of security protocols has been proved to be undecidable in general, several approaches use simplifying hypotheses in order to obtain decidability for interesting subclasses. Amongst the most common is type abstraction, *i.e.* considering only well-typed runs of the protocol, therefore bounding message length. In this paper, we show how to get message boundedness "for free" under a reasonable (syntactic) assumption on protocols, in order to verify a variety of interesting security properties including secrecy and several authentication properties. This enables us to improve existing decidability results by restricting the search space for attacks.

Keywords: Cryptographic protocols, Formal methods, Verification, Secrecy, Authentication

1. Introduction

Security protocols are short programs that describe communication between two or more parties in order to achieve security goals such as data confidentiality or identification of a correspondent for example. These protocols are executed in a hostile environment, such as the Internet, and aim at preventing a malicious agent from tampering with the messages, for instance, using encryption. However, encrypting messages is not sufficient to enforce security or privacy guarantees. History has shown that these protocols are extremely error-prone and careful formal verification is needed.

Despite the apparent simplicity of such protocols, their verification is a difficult problem and has been proved undecidable in general [DLMS99, CC01]. Different attempts [DLMS99, Low99, RT01, CC03, RS03a, RS03b, BP05] have successfully exhibited decidable subclasses, and tools for proving security have been designed [Mea96, CJM00, Bla01, CMR01, SBP01]. However, termination

*Corresponding author
Email addresses: marapini@inf.ed.ac.uk (Myrto Arapinis), marie.duflot-kremer@loria.fr (Marie Duflot)

of these tools is often not guaranteed. In the literature, only very few results consider an unbounded number of sessions and even fewer allow unbounded creation of new nonces (constants generated to ensure freshness of messages). Indeed, most existing decidability results were obtained either under the abstraction that only a bounded number of nonces are ever created, or considering only a bounded number of sessions of the protocol (see the recapitulative tables pages 6 and 10 in [CDL06]).

In order to obtain decidability, many existing results bound the message length in adopting a typing abstraction [Low99, RS03a, DLMS99, CKR$^+$03], an assumption according to which one can always tell the type of a given message. While this appears as an unrealistic assumption in the general case, this paper presents a simple way of justifying it. This question has been addressed in [LYH04, HLS03] amongst others, and solved with tagging schemes.

In this paper, we show that when a protocol satisfies a reasonable syntactic condition based on non-unifiability of subterms of different types - and belongs to a class we will denote $\mathbb{P}$, then the typing abstraction is correct for several security properties, including for example secrecy and several forms of authentication. Furthermore, the considered typing system here is much more fine-grained than the one considered in [LYH04, HLS03], thus refining existing results by reducing the search space to be handled. Indeed, we prove that a protocol in our class $\mathbb{P}$ admits an attack if and only if it admits a "well-typed" attack with respect to a stronger notion of typing.

Our class of protocols characterizes protocols in which an encrypted term cannot be mistaken for another, unless they are of the same type. This notion is often satisfied in protocols found in the literature (see [CJ97]) and, even when the protocol is not well-formed, a light tagging like the one proposed in [BP05] permits one to comply with the property and thus use our result.

This paper is a revised and extended version of [AD07]. Unlike the previous one, this work tackles a larger class of security properties. They are now defined using the logic $\mathcal{PS}$-LTL and characterized by a newly defined class that includes not only secrecy but other interesting security properties related to authentication and fairness. We also present extensive proofs of the results.

The rest of the paper is organized as follows. Section 2 presents the formalism used throughout the paper, both for the description of the content of a message, and for the capabilities of the intruder to decrypt messages and forge new ones. Section 3 gives the model we consider for security protocols. In Section 4 we present the logic used to describe the security properties to verify. We also explain how to express, using this logic, several well known security properties. Section 5 is dedicated to the statement, first informally and then in more details of the main result of this paper, introducing in particular the typing system considered.Then the formal proof together with the tools needed to write it can be found in Section 6. Section 7 presents applications of this result, as well as a discussion on the (un)decidability of the secrecy problem for the class of protocols $\mathbb{P}$. More precisely, we show that the number of sessions to consider to find an attack on a protocol in $\mathbb{P}$ cannot be bounded in advance by a constant. We finally conclude in Section 8. In order to help the reader keeping track of

all notations and technical terms used in the paper, an index can be found just after the bibliography.

Related work. Our work presents a reduction result of the search space to well-typed runs for the class of protocols satisfying the criterion of "non-unifiability of sub-terms" stated in [AN96]. Many works take this reduction result as an assumption to prove decidability results, such as [Low99, RS03a, DLMS99, CKR+03, HS06]. However, such an abstraction is not correct in general (this can be seen by the attack present in the protocol Π_{Toy} that we use as running example here).

The closest work to ours is the one of J. Heather *et al.* [HLS03], where the authors also present a reduction result of the search space to well-typed runs. Indeed, they show that it is possible to label the terms of a protocol with static tags such that the typing assumption is correct. However, their result does not apply to protocols involving blind copies, *i.e.* variables with complex types, nor complex keys. It doesn't either apply to protocols that do not use static tags. Our result is more general as it applies to protocols that satisfy non-unifiability of subterms of different types, but not necessarily through the use of static tags. Moreover, the reduction result presented in [HLS03] is more coarse grained, as all nonces have the same type.

2. Messages and intruder capabilities

This section is dedicated first to the presentation of the formalism used to describe the considered cryptographic primitives and messages. We then precisely characterize the intruder capabilities, that is how the intruder can decompose messages received in order to gain information, as well as how he can construct new messages in order to fool the other participants.

2.1. Messages

We use an abstract term algebra to model the messages of a protocol. For this we fix several disjoint sets. We consider an infinite set of *agents* $\mathcal{A} = \{\epsilon, a, b \ldots\}$ with the special agent ϵ standing for the attacker, and an infinite set of *agent variables* $\mathcal{Z} = \{z_A, z_B, \ldots\}$. Agent variables in $\mathcal{Z}$ will denote the roles of the protocol. We also need to assume an infinite set of *honest names* $\mathcal{N} = \{n, m \ldots\}$ to model atomic data such as nonces, an infinite set of *dishonest names* $\mathcal{N}^\epsilon = \{m^\epsilon, n^\epsilon, n^{\epsilon,1}, n^{\epsilon,2} \ldots\}$ standing for names generated by the attacker, and an infinite set of *term variables* $\mathcal{X} = \{x, y, x_1, x_2, \ldots\}$. In what follows, we will often refer to term variables simply as variables, in order to avoid unnecessary and cumbersome repetitions. We consider an infinite set of constants $\mathcal{C} = \{c, d, c_1, c_2, \ldots\}$ and the following *signature* $\mathcal{F} = \{\mathsf{senc}/2, \mathsf{aenc}/2, \mathsf{sign}/2, \langle\,\rangle/2, \mathsf{h}/1, \mathsf{pv}/1, \mathsf{sh}/2\}$. These function symbols model cryptographic primitives. The symbol $\langle\,\rangle$ represents pairing. The term $\mathsf{senc}(m, k)$ (*resp.* $\mathsf{aenc}(m, k)$) represents the message m encrypted with the symmetric (resp. asymmetric) key k whereas the term $\mathsf{sign}(m, k)$ represents the message m signed by the key k. The function h models

a hash function whereas $\mathsf{pv}(a)$ is used to model the private key of agent a, and $\mathsf{sh}(a, b)$ is used to model the long-term symmetric key shared by agents a and b. In order to avoid reasoning modulo the equational theory $\mathsf{sh}(z_1, z_2) = \mathsf{sh}(z_2, z_1)$, we equip the set $\mathcal{A}$ with a total order denoted $<_{\mathcal{A}}$, and will only allow keys $\mathsf{sh}(a, b)$ with $a <_{\mathcal{A}} b$. We assume ϵ to be such that for all $a \in \mathcal{A}$, $\epsilon <_{\mathcal{A}} a$. The set of *terms* $\mathcal{T}$ is defined inductively by the following grammar:

$t, t_1, t_2, \ldots$	::=		term
	|	a	agent $a \in \mathcal{A}$
	|	z	agent variable $z \in \mathcal{Z}$
	|	n	name $n \in \mathcal{N} \cup \mathcal{N}^{\epsilon}$
	|	x	term variable $x \in \mathcal{X}$
	|	c	constant $c \in \mathcal{C}$
	|	$\mathsf{pv}(u)$	application of the symbol pv on $u \in \mathcal{A} \cup \mathcal{Z}$
	|	$\mathsf{sh}(u_1, u_2)$	application of the symbol sh on $u_1, u_2 \in \mathcal{A} \cup \mathcal{Z}$
	|	$\mathsf{h}(t)$	application of h
	|	$\mathsf{f}(t_1, t_2)$	application of symbol $\mathsf{f} \in \{\mathsf{senc}, \mathsf{aenc}, \mathsf{sign}, \langle\,\rangle\}$

We say that a term is *ground* if no agent or term variable appears in it. We add to the above grammar the restriction that any ground shared key $\mathsf{sh}(a, b)$ with $a, b \in \mathcal{A}$ satisfies $a <_{\mathcal{A}} b$. We consider the usual notations for manipulating terms. We write $\mathcal{X}(t)$ (resp. $\mathcal{N}(t)$, $\mathcal{A}(t)$, $\mathcal{Z}(t)$, $\mathcal{C}(t)$) for the set of term variables (resp. names, agents, agent variables, and constants) occurring in t. We write $\mathsf{St}(t)$ for the set of *subterms* of a term t. Note that in our convention a key is a subterm of an encrypted message. We then define the set of *cryptographic subterms* of a term t as $\mathsf{CryptSt}(t) = \{\mathsf{f}(t_1, \ldots, t_n) \in \mathsf{St}(t) \mid \mathsf{f} \in \{\mathsf{senc}, \mathsf{aenc}, \mathsf{sign}, \mathsf{h}\}\}$.

Moreover we define $\mathcal{K}^{\epsilon} = \{\mathsf{pv}(\epsilon)\} \cup \{\mathsf{sh}(\epsilon, a) \mid a \in \mathcal{A}\}$. Intuitively $\mathcal{K}^{\epsilon}$ represents the set of long-term keys of the attacker. An *atom* is a long-term key, an agent name, a name or an agent or term variable.

Sometimes we'll need to refer to the set of subterms not occurring only in key position in t. For this reason, we define the set of *plaintexts* of a term t as the set of atoms that occur in plaintext, *i.e*

- $\mathit{plaintext}(\mathsf{h}(u)) = \mathit{plaintext}(\mathsf{f}(u, v)) = \mathit{plaintext}(u)$ for $\mathsf{f} \in \{\mathsf{senc}, \mathsf{aenc}, \mathsf{sign}\}$,
- $\mathit{plaintext}(\langle u, v\rangle) = \mathit{plaintext}(u) \cup \mathit{plaintext}(v)$, and
- $\mathit{plaintext}(u) = \{u\}$ otherwise.

All these notions are extended to sets of terms and to other kinds of term containers as expected.

A substitution is written $\sigma = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ where its *domain* is $\mathrm{dom}(\sigma) = \{x_1, \ldots, x_n\} \in (\mathcal{X} \cup \mathcal{Z})^n$. The substitution σ is *ground* if all the t_i's are ground. The application of a substitution σ to a term t is written $\sigma(t)$ or $t\sigma$. Two terms t_1 and t_2 are *unifiable* if $t_1\sigma = t_2\sigma$ for some substitution σ, that is called a *unifier*; σ is a *most general unifier* if for any other unifier θ of t_1 and t_2 there exists a substitution θ' such that $t_1\theta = t_1\sigma\theta' = t_2\theta = t_2\sigma\theta'$. The *most general unifier* is unique up to agent and term variable renaming. We denote

by $\mathrm{mgu}(t_1, t_2)$ the *most general unifier* of t_1 and t_2 and write $\mathrm{mgu}(t_1, t_2) = \bot$ if t_1 and t_2 are not unifiable. An algorithm for computing the mgu will be given in Section 6.1.1.

Example 1. *Let* $t = \mathsf{senc}(\langle n, a\rangle, \mathsf{sh}(a, b))$. *We have that* $\mathcal{X}(t) = \mathcal{Z}(t) = \emptyset$, *i.e.* t *is ground,* $\mathcal{N}(t) = \{n\}$, $\mathsf{St}(t) = \{t, \langle n, a\rangle, \mathsf{sh}(a, b), n, a, b\}$, *plaintext*$(t) = \{n, a\}$ *and* $\mathcal{A}(t) = \{a, b\}$. *The terms* $\mathsf{sh}(a, b)$, a, b *and* n *are atoms.*

2.2. Intruder capabilities

We model the intruder's ability to construct new messages by the deduction system given in Figure 1. The intuitive meaning of these rules is that an intruder can compose new messages by pairing, encrypting, signing and hashing previously known messages, provided he has the corresponding keys. Conversely, he can decompose messages by projecting or decrypting provided he has the corresponding decryption keys. As can be seen in the rules and in the grammar characterizing terms, we associate the encryption key of one agent to his name. Indeed $\mathsf{aenc}(u, v)$ denotes the encryption of term u using v's public key, and $\mathsf{pv}(v)$ is the corresponding decryption key. Note that while we allow asymmetric encryption under any term, we only allow decryption using a key of the form $\mathsf{pv}(u)$ with $u \in \mathcal{A} \cup \mathcal{Z}$. This amounts to associating the public key of participants to their identity. For a symmetric encryption $\mathsf{senc}(u, v)$, v denotes the key, used for both encryption and decryption. Our optional rule expresses that an intruder can retrieve the whole message from its signed version. Whether this property holds depends on the actual signature scheme. Therefore we consider this rule to be optional. Our results hold in both cases.

$$\frac{u \quad v}{\langle u, v\rangle} \qquad \frac{u \quad v}{\mathsf{senc}(u, v)} \qquad \frac{u \quad v}{\mathsf{aenc}(u, v)} \qquad \frac{u \quad v}{\mathsf{sign}(u, v)} \qquad \frac{u}{\mathsf{h}(u)}$$

$$\frac{\langle u, v\rangle}{u} \qquad \frac{\langle u, v\rangle}{v} \qquad \frac{\mathsf{senc}(u, v) \quad v}{u} \qquad \frac{\mathsf{aenc}(u, v) \quad \mathsf{pv}(v)}{u} \qquad \frac{\mathsf{sign}(u, v)}{u} \text{ (optional)}$$

Figure 1: Intruder deduction system.

Definition 1 (deducible). We say that a term u is *deducible* from a set of terms T, denoted $T \vdash u$, if there exists a tree such that its root is labelled by u, its leaves are labelled with $v \in T \cup \mathcal{A} \cup \mathcal{K}^{\epsilon} \cup \mathcal{N}^{\epsilon} \cup \mathcal{C}$, and for every node labelled by v having n sons labelled by $v_1, \ldots, v_n$ we have that $\dfrac{v_1 \ \ldots \ v_n}{v}$ is an instance of one of the inference rules given in Figure 1.

Example 2. *Given* $T_1 \stackrel{def}{=} T_0 \cup \{\mathsf{aenc}(\langle \mathsf{aenc}(n^1, b), a\rangle, b)\}$ *where* T_0 *is any set of terms, we have* $T_1 \vdash \mathsf{aenc}(\langle \mathsf{aenc}(\langle \mathsf{aenc}(n^1, b), a\rangle, b), \epsilon\rangle, b)$ *according to the following tree.*

$$\dfrac{\dfrac{\mathsf{aenc}(\langle \mathsf{aenc}(n^1, b), a\rangle, b) \qquad \epsilon}{\langle \mathsf{aenc}(\langle \mathsf{aenc}(n,^1 b), a\rangle, b), \epsilon\rangle} \qquad b}{\mathsf{aenc}(\langle \mathsf{aenc}(\langle \mathsf{aenc}(n^1, b), a\rangle, b), \epsilon\rangle, b)}$$

Now if $T_2 \stackrel{def}{=} T_1 \cup \{\mathsf{aenc}(\langle \mathsf{aenc}(\langle \mathsf{aenc}(n^1, b), a\rangle, \epsilon), b\rangle, \epsilon)\}$*, we have* $T_2 \vdash \mathsf{aenc}(\langle \mathsf{aenc}(n^1, b), \epsilon\rangle, b)$ *according the tree.*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathsf{aenc}(\langle \mathsf{aenc}(\langle \mathsf{aenc}(n^1, b), a\rangle, \epsilon), b\rangle, \epsilon) \qquad \mathsf{pv}(\epsilon)}{\langle \mathsf{aenc}(\langle \mathsf{aenc}(n^1, b), a\rangle, \epsilon), b\rangle}}{\mathsf{aenc}(\langle \mathsf{aenc}(n^1, b), a\rangle, \epsilon)} \qquad \mathsf{pv}(\epsilon)}{\langle \mathsf{aenc}(n^1, b), a\rangle}}{\mathsf{aenc}(n^1, b)} \qquad \epsilon}{\langle \mathsf{aenc}(n^1, b), \epsilon\rangle} \qquad b}{\mathsf{aenc}(\langle \mathsf{aenc}(n^1, b), \epsilon\rangle, b)}$$

And finally if $T_3 \stackrel{def}{=} T_2 \cup \{\mathsf{aenc}(\langle \mathsf{aenc}(n^1, \epsilon), b\rangle, \epsilon)\}$*, the following tree shows that* $T_3 \vdash n^1$*.*

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathsf{aenc}(\langle \mathsf{aenc}(n^1, \epsilon), b\rangle, \epsilon) \qquad \mathsf{pv}(\epsilon)}{\langle \mathsf{aenc}(n^1, \epsilon), b\rangle}}{\mathsf{aenc}(n^1, \epsilon)} \qquad \mathsf{pv}(\epsilon)}{n^1}$$

3. Modeling security protocols

3.1. Syntax

We consider protocols specified in a language allowing parties to exchange messages built from identities and randomly generated nonces using public and symmetric key encryption and digital signatures. The individual behavior of each protocol participant is defined by a *role* describing a sequence of *events*. The main events we consider are *communication events*, *i.e.* message receptions and message transmissions. To be able to specify a large class of security properties (a logic of properties is given in Section 4), we also consider *status events*. Those events are issued by participants to mark different stages reached in the execution of a protocol role.

Definition 2 (event). An *event* is one of the following:

- a *communication event*, i.e. a message reception denoted by $\mathsf{rcv}(r_A, r_B, m)$ or a message transmission denoted by $\mathsf{snd}(r_A, r_B, m)$, where m is a term, and $r_A, r_B \in \mathcal{Z} \cup \mathcal{A}$; or

- a *status event* of the form $\mathsf{Q}(r, t_1, \ldots, t_n)$ where each t_i is a term (not necessarily ground), $r \in \mathcal{Z} \cup \mathcal{A}$ is the participant who generated the event, and Q is a predicate symbol of arity $n+1$.

Typically, a status event gives information about the state of the principal that generated it. For instance, we will consider status events that indicate that the principal has started or finished an execution, and possibly with whom. The set of agent and term variables of an event is defined as expected, considering all the terms occurring in the event's specification.

We will write $i \in [\![n]\!]$ for $i \in \{1, \ldots, n\}$. Appending an event e to a sequence of events seq is written $seq; \mathsf{e}$, and the concatenation of two sequences seq_1 and seq_2 is written $seq_1 @ seq_2$. The function length has the usual meaning: $\mathsf{length}([\,]) = 0$ and $\mathsf{length}(seq; \mathsf{e}) = 1 + \mathsf{length}(seq)$. The prefix sequence consisting of the first i events is denoted as seq_i, with $seq_0 = [\,]$ and $seq_n = seq$ when $n \geq \mathsf{length}(seq)$.

We will denote $\mathsf{Comms}(seq)$, the restriction of the sequence of events seq to its communication events, and $seq(r)$ the restriction of seq to the events initiated by $r \in \mathcal{Z} \cup \mathcal{A}$, introduced in the following definition.

Definition 3. Given a sequence of events seq and an agent name or variable $p_1 \in \mathcal{Z} \cup \mathcal{A}$, the restriction of seq to the actions initiated by p_1 is formally defined by:

$$seq(p_1) = \begin{cases} [] & \text{if } seq = [] \\ [e]@seq'(p_1) & \text{if } seq = [e]@seq' \text{ and } \quad e = \mathsf{snd}(p_1, p_2, t) \\ & \qquad\qquad\qquad\quad \text{or} \quad e = \mathsf{rcv}(p_1, p_2, t) \\ & \qquad\qquad\qquad\quad \text{or} \quad e = \mathsf{Q}(p_1, t_1, \ldots t_n) \\ seq'(p_1) & \text{otherwise} \qquad\qquad\quad \text{with } seq = [e]@seq' \end{cases}$$

Definition 4 (Protocol). A protocol is a sequence of events $\Pi = [e_1; \ldots; e_\ell]$ satisfying the following conditions.

1. $\forall r_A, r_B \in \mathcal{Z} \cup \mathcal{A},\ r_A \neq r_B \ \Rightarrow$
$$\mathcal{N}(\Pi(r_A)) \cap \mathcal{N}(\Pi(r_B)) = \emptyset \ \wedge\ \mathcal{X}(\Pi(r_A)) \cap \mathcal{X}(\Pi(r_B)) = \emptyset$$
2. $\mathsf{Comms}(\Pi)$ is of the form $[e'_1; e''_1; \ldots; e'_m; e''_m]$, such that for all $i \in [\![m]\!]$, $e'_i = \mathsf{snd}(r_i, r'_i, u_i)$ and $e''_i = \mathsf{rcv}(r'_i, r_i, v_i)$, and
   (a) if $x \in \mathcal{X}(u_i)$, then there exists $j \in [\![i-1]\!]$ such that $x \in \mathcal{X}(v_j)$,
   (b) there exists a tuple of substitutions $(\delta_1, ..., \delta_m)$ with $\delta_i \neq \bot$ and
$$\begin{cases} \delta_1 = \mathrm{mgu}(u_1, v_1), \\ \delta_k = \mathrm{mgu}(u_k\delta_1 \ldots \delta_{k-1}, v_k\delta_1 \ldots \delta_{k-1}) \end{cases}$$
   The substitution $\delta_\Pi = \delta_1 \ldots \delta_m$ is called the *honest substitution* of Π.
3. $\forall i \in [\![\ell]\!]$, if $e_i = \mathsf{Q}(r_A, t_1, \ldots, t_n)$, then
$\forall j \in [\![n]\!],\ \forall x \in \mathcal{X}(t_j),\ \exists r_B \in \mathcal{Z} \cup \mathcal{A},\ \exists k \in [\![i]\!],$
$$e_k = \mathsf{rcv}(r_A, r_B, t) \ \wedge\ x \in \mathcal{X}(t)$$

4. $r_i, r'_i \in \mathcal{Z} \cup \mathcal{A}$
5. $\forall z \in \mathcal{Z}(\Pi),\ \exists i \in [\![\ell]\!]$ such that
$$e_i = \mathsf{snd}(z, z', t) \ \vee \ e_i = \mathsf{rcv}(z, z', t) \ \vee \ e_i = \mathsf{Q}(z, t_1, \ldots, t_n)$$

The set of participants of a protocol is the set $\mathsf{Partcpts}(\Pi) = \{r \in \mathcal{Z} \cup \mathcal{A} \mid \Pi(r) \neq []\}$. The restriction of a protocol to the actions initiated by one of its participants is called a *role*.

The first condition states that, in a protocol specification, each nonce appears in the events initiated by at most one participant of the protocol. This corresponds to the fact that each nonce is generated by a unique participant. This also holds for the term variables occurring in the specification of the protocol. Condition 2 restricts protocols to sequences of events in which each emission corresponds to a reception (and *vice versa*). This allows in particular to exclude a certain number of "protocols" that don't have a "normal execution" (*i.e.* with only honest participants following the protocol). Conditions 3 and 2a combined with Condition 1 ensure that a term variable is used in a status event or an emission initiated by a participant only if he has indeed previously received the particular term variable. Indeed, Condition 1 enforces that $r_i = r'_j$ in Condition 2a. Condition 3 is often referred to in the literature as the *origination property*. The last two conditions ensure that all the agent variables in the protocol specification will be instantiated by agent names in any symbolic trace and thus in any execution of the protocol.

Example 3. *As an illustrative example we will consider all along this paper the following toy protocol* Π_{Toy} *that can be informally described as follows.*

$$\begin{array}{lcll} a & \to & b: & \{\{n\}_b, a\}_b \\ b & \to & a: & \{\{n\}_a, b\}_a \end{array}$$

where $\{m\}_c$ *denotes encryption of message* m *with* c*'s public key. This protocol is composed of two messages exchanged between participants* a *and* b*. First* a *generates the secret* n*, encrypts it with* b*'s public key, concatenates his identity, and encrypts with* b*'s public key before sending to* b *the whole message.* b *acknowledges the receipt of* a*'s message by decrypting it twice with his private key, re-encrypting with* a*'s public key, and including his own identity. Using the previously introduced formalism, this protocol corresponds to the following sequence of events.*

$$\Pi_{\mathsf{Toy}} \ = \ \left[\begin{array}{l} \mathsf{snd}(z_a, z_b, \mathsf{aenc}(\langle \mathsf{aenc}(n, z_b), z_a \rangle, z_b)); \\ \mathsf{rcv}(z_b, z_a, \mathsf{aenc}(\langle \mathsf{aenc}(x, z_b), z_a \rangle, z_b)); \\ \mathsf{snd}(z_b, z_a, \mathsf{aenc}(\langle \mathsf{aenc}(x, z_a), z_b \rangle, z_a)); \\ \mathsf{rcv}(z_a, z_b, \mathsf{aenc}(\langle \mathsf{aenc}(n, z_a), z_b \rangle, z_a)) \end{array}\right]$$

It illustrates in particular that one message sent in a protocol corresponds to two events: the sending of the message and the expected reception of this message. Following Definition 4, the honest substitution of Π_{Toy} *is* $\delta_{\Pi_{\mathsf{Toy}}} = \{x \mapsto n\}$*. This*

protocol has two participants z_a and z_b and their respective roles are described below.

$$\Pi_{\mathsf{Toy}}(z_a) = \left[\begin{array}{l} \mathsf{snd}(z_a, z_b, \mathsf{aenc}(\langle \mathsf{aenc}(n, z_b), z_a\rangle, z_b)); \\ \mathsf{rcv}(z_a, z_b, \mathsf{aenc}(\langle \mathsf{aenc}(n, z_a), z_b\rangle, z_a)) \end{array}\right]$$

$$\Pi_{\mathsf{Toy}}(z_b) = \left[\begin{array}{l} \mathsf{rcv}(z_b, z_a, \mathsf{aenc}(\langle \mathsf{aenc}(x, z_b), z_a\rangle, z_b)); \\ \mathsf{snd}(z_b, z_a, \mathsf{aenc}(\langle \mathsf{aenc}(x, z_a), z_b\rangle, z_a)) \end{array}\right]$$

3.2. Semantics

In our model, a session corresponds to the instantiation of one role. This means in particular that one "normal execution" of a k-party protocol requires k sessions (one per role). We may want to consider several sessions corresponding to different instantiations of a same role. Since the adversary may block, redirect and send new messages, the sessions might be interleaved in many ways. Such an interleaving is captured by the notion of *scenario*.

Definition 5 (scenario). A *scenario for a k-party protocol* Π is a couple $\mathsf{sc} = (intl, \alpha)$ with the considered interleaving $intl$ and instantiation for agent variables $\alpha : (\mathbb{N} \times \mathcal{Z}) \to \mathcal{A}$. The sequence $intl = [(r_1, sid_1); \cdots ; (r_n, sid_n)]$ is such that r_i is a participant of Π, sid_i is a session identifier such that for every i, $sid_i \in \mathbb{N}$, the number of identical occurrences of a pair (r, sid) is at most the length of the role $\Pi(r)$, and $sid_i = sid_j$ implies $r_i = r_j$. The function α is such that for all $i \in [\![n]\!]$ and for all $r \in \mathcal{Z}(\Pi)$, $\alpha(sid_i, r)$ is defined.

The condition on identical occurrences ensures that a session consists of no more events than specified by the corresponding role. The last condition ensures that a session number corresponds precisely to one instance of one role. We say that $(r, s) \in \mathsf{sc}$ if (r, s) is an element of the sequence $intl$. The last condition ensures that all agent variables will be instantiated by an agent identity in the corresponding symbolic trace (Definition 6).

Given a scenario, we define a *symbolic trace*, that is the sequence of events corresponding to the interleaving of the scenario, for which the agent variables have been instantiated by agent names, fresh nonces are generated and term variables are renamed to avoid name collisions between different sessions.

Definition 6 (symbolic trace). Let Π be a k-party protocol with a set of participants $\mathsf{Partcpts}(\Pi) = \{r_1, \ldots, r_k\}$ and respective roles $\Pi(r_i) = [\mathsf{e}^i_1; \ldots ; \mathsf{e}^i_{\ell_i}]$ for $1 \leq i \leq k$.

Given a scenario $\mathsf{sc} = ([(r_1, sid_1); \cdots ; (r_n, sid_n)], \alpha)$, the *symbolic trace* $\mathsf{tr} = [\mathsf{e}_1; \ldots ; \mathsf{e}_n]$ *associated to* sc is defined as follows. Let $q_i = \#\{(r_j, sid_j) \in \mathsf{sc} \mid j \leq i, sid_j = sid_i\}$, i.e. the number of occurrences to this point in sc of the session sid_i. We have $q_i \leq \mathsf{length}(\Pi(r_i))$ and $\mathsf{e}_i = \mu_{r_i, sid_i}(\mathsf{e}^i_{q_i})$, where $\mu_{r,sid}$ is a function from terms to terms

- $\mu_{r,sid}(n) = n^{sid}$ if $n \in \mathcal{N}(\Pi)$, where n^{sid} is a fresh term name in $\mathcal{N}$;
- $\mu_{r,sid}(z) = \alpha(sid, z)$ if $z \in \mathcal{Z}(\Pi)$;

- $\mu_{r,sid}(x) = x^{sid}$ if $x \in \mathcal{X}(\Pi)$, where x^{sid} is a fresh variable in $\mathcal{X}$.
- $\mu_{r,sid}(\mathsf{sh}(p_1, p_2)) = \mathsf{sh}(\mu_{r,sid}(p_i), \mu_{r,sid}(p_j))$, where $\{i, j\} = \{1, 2\}$ and $\mu_{r,sid}(p_i) <_{\mathcal{A}} \mu_{r,sid}(p_j)$;
- $\mu_{r,sid}(u) = u$ if $u \in \mathcal{C} \cup \mathcal{A}$;
- $\mu_{r,sid}(\mathsf{f}(u_1, \ldots, u_n)) = \mathsf{f}(\mu_{r,sid}(u_1), \ldots, \mu_{r,sid}(u_n))$, where $\mathsf{f} \in \mathcal{F}$ but $\mathsf{f} \neq \mathsf{sh}$.

A session *sid* is said to be *honest* w.r.t. α when $\forall z \in \mathcal{Z}(\Pi), \alpha(sid, z) \in \mathcal{A} \setminus \{\epsilon\}$.

Intuitively, a session *sid* is honest if all of its participants, from the point of view of the agent executing session *sid*, are honest (i.e. $\neq \epsilon$). Note that a symbolic trace does not contain agent variables, but may of course contain term variables in $\mathcal{X}$.

Example 4. $\mathsf{sc}_{\mathsf{Toy}} = (intl, \alpha)$, *with*

- *the interleaving* $intl = [(z_a, 1); (z_b, 2); (z_b, 2); (z_b, 3); (z_b, 3)]$, *and*
- *the instantiation function* α *defined by* $\alpha(1, z_a) = a$, $\alpha(1, z_b) = \alpha(2, z_b) = \alpha(3, z_b) = b$ *and* $\alpha(2, z_a) = \alpha(3, z_a) = \epsilon$,

is a scenario for the protocol of Example 3 page 8. This scenario engages 3 sessions: session 1 initiated by agent a, and sessions 2 and 3 initiated by agent b. In session 1, agent a wants to execute role z_a *with b as* z_b*. In sessions 2 and 3, agent b wants to execute role* z_b *with the intruder* ϵ *as* z_a*. The following symbolic trace* $\mathsf{tr}_{\mathsf{Toy}}$ *corresponds to* $\mathsf{sc}_{\mathsf{Toy}}$.

$$\mathsf{tr}_{\mathsf{Toy}} = \begin{bmatrix} \mathsf{snd}(a, b, \mathsf{aenc}(\langle \mathsf{aenc}(n^1, b), a \rangle, b)); \\ \mathsf{rcv}(b, \epsilon, \mathsf{aenc}(\langle \mathsf{aenc}(x^2, b), \epsilon \rangle, b)); \\ \mathsf{snd}(b, \epsilon, \mathsf{aenc}(\langle \mathsf{aenc}(x^2, \epsilon), b \rangle, \epsilon)); \\ \mathsf{rcv}(b, \epsilon, \mathsf{aenc}(\langle \mathsf{aenc}(x^3, b), \epsilon \rangle, b)); \\ \mathsf{snd}(b, \epsilon, \mathsf{aenc}(\langle \mathsf{aenc}(x^3, \epsilon), b \rangle, \epsilon)) \end{bmatrix}$$

An *execution trace* is a ground instance of such a symbolic trace. As usual, we are only interested in *valid* execution traces - those traces where the attacker only sends messages that he can actually compute from his initial knowledge and the messages he has seen on the network so far.

Definition 7 (knowledge of a trace tr). Let tr be a trace. The *knowledge* gained by the intruder with tr is the set of terms given by the following formula.

$$\mathsf{K}(\mathsf{tr}) = \{u \mid \mathsf{snd}(r, r', u) \in \mathsf{tr}\}$$

Definition 8 (valid execution trace). Let T_0 be a finite set of ground terms (intuitively T_0 represents the initial knowledge of the attacker). A ground execution $\mathsf{exec} = [\mathsf{e}_1; \ldots; \mathsf{e}_\ell]$ is *valid* w.r.t. T_0 if for all $1 \leq i \leq \ell$, whenever $\mathsf{e}_i = \mathsf{rcv}(a, b, m)$, we have that $T_0 \cup \mathsf{K}(\mathsf{exec}_{i-1}) \vdash m$.

Example 5. *Reusing the symbolic trace of Example 4 and applying the substitution* $\sigma = \{x^2 \mapsto \langle \mathsf{aenc}(n^1, b), a\rangle; x^3 \mapsto n^1\}$ *we obtain a valid execution. The intruder can indeed, given his knowledge at that time of the trace, create every message that is received by any agent, as illustrated in the two first derivations of Example 2 page 5.*

4. Security properties

This section is dedicated to the presentation of the logic we will use in the remaining of this paper. We first present $\mathcal{PS}$-LTL, a linear time logic with past modalities presented in [CES06] and then illustrate the expressivity of this logic by formalizing several security properties in $\mathcal{PS}$-LTL.

4.1. $\mathcal{PS}$-LTL

Definition 9. A formula of $\mathcal{PS}$-LTL is an expression ϕ defined by the following grammar

$$\begin{array}{rcl} \phi & ::= & \mathsf{true} \mid \mathsf{Q}(t_1, \ldots, t_n) \mid \mathsf{learn}(t_1) \mid \mathsf{C}(t_1) \\ & & \mid \neg\phi \mid \phi \vee \phi \mid \mathcal{Y}\phi \mid \phi \mathcal{S} \phi \mid \exists x, \phi \end{array}$$

where $t_1, \ldots, t_n$ are terms and $x \in \mathcal{X}$.

In the sequel, we assume that formulas are *closed*, i.e. they contain no free variables, and that each variable is quantified at most once (this can be easily ensured by renaming). We also assume that the variables occurring in a formula ϕ are disjoint from the variables occurring in the considered symbolic trace tr.

Formulas are interpreted at some position along an execution as stated in Definition 10.

Definition 10 (concrete validity). Let ϕ be a closed $\mathcal{PS}$-LTL formula, exec be an execution and T_0 be a finite set of ground terms. We define $\langle \mathsf{exec}, T_0\rangle \models \phi$ as:

$$\begin{array}{lll} \langle \mathsf{exec}, T_0\rangle \models \mathsf{true} & & \\ \langle \mathsf{exec}, T_0\rangle \models \mathsf{learn}(m) & \text{iff} & T_0 \cup \mathsf{K}(\mathsf{exec}) \vdash m \\ \langle \mathsf{exec}, T_0\rangle \models \neg\phi & \text{iff} & \langle \mathsf{exec}, T_0\rangle \not\models \phi \\ \langle \mathsf{exec}, T_0\rangle \models \phi_1 \vee \phi_2 & \text{iff} & \langle \mathsf{exec}, T_0\rangle \models \phi_1 \text{ or } \langle \mathsf{exec}, T_0\rangle \models \phi_2 \\ \langle \mathsf{exec}, T_0\rangle \models \exists x, \phi & \text{iff} & \text{there exists a ground term } t \text{ such that} \\ & & \qquad \langle \mathsf{exec}, T_0\rangle \models \phi\{x \mapsto t\} \\ \langle \mathsf{exec}, T_0\rangle \models \mathsf{Q}(t_1, \ldots, t_n) & \text{iff} & \mathsf{exec} = \mathsf{exec}'; \mathsf{Q}(t_1, \ldots, t_n) \\ \langle \mathsf{exec}, T_0\rangle \models \mathsf{C}(u) & \text{iff} & T_0 \vdash \mathsf{pv}(u) \text{ or } T_0 \vdash \mathsf{sh}(u, v) \text{ for some } v \neq \epsilon \\ \langle \mathsf{exec}, T_0\rangle \models \mathcal{Y}\phi & \text{iff} & \mathsf{exec} = \mathsf{exec}'; e \text{ and } \langle \mathsf{exec}', T_0\rangle \models \phi \\ \langle \mathsf{exec}, T_0\rangle \models \phi_1 \mathcal{S} \phi_2 & \text{iff} & \exists i \in [0, \mathsf{length}(\mathsf{exec})] \text{ such that } \langle \mathsf{exec}_i, T_0\rangle \models \phi_2 \\ & & \quad \text{and } \forall j \in [i+1, \mathsf{length}(\mathsf{exec})], \langle \mathsf{exec}_j, T_0\rangle \models \phi_1 \end{array}$$

While most of these operators are very usual, we can here informally give the meaning of the more specific ones. $\mathsf{learn}(m)$ means that the intruder can deduce the term m from his current knowledge. $\mathsf{C}(u)$ denotes the fact that agent u is compromised, *i.e.* either u is the intruder or he is not but the intruder knows his private key or a key u shares with another "honest" agent. We will use in the following the notation $\mathsf{NC}(u)$ for $\neg\mathsf{C}(u)$. Last, the temporal operator $\mathcal{Y}\phi$ (yesterday) means that ϕ was true at the previous step, and $\phi_1\mathcal{S}\phi_2$ that ϕ_2 held some time in the past and that ϕ_1 has been true ever since. We use the notation $\Diamond\phi$, also called "once" as a shortcut for $\mathsf{true}\,\mathcal{S}\phi$. Furthermore we use the usual notations $\forall x, \phi$ for $\neg\exists x, \neg\phi$ and $\phi_1 \wedge \phi_2$ for $\neg((\neg\phi_1) \vee (\neg\phi_2))$.

In the following, all the security properties we will consider will belong to a particular subclass of $\mathcal{PS}$-LTL, namely $\mathcal{PS}$-LTL$^-$. In order to introduce this subclass we first need to define positive and negative subformulas.

$$\mathsf{Sf}^-(\phi) \stackrel{def}{=} \begin{cases} \mathsf{Sf}^+(\phi_1) & \text{if } \phi = \neg\phi_1 \\ \mathsf{Sf}^-(\phi_1) \cup \mathsf{Sf}^-(\phi_2) & \text{if } \phi = \phi_1 \vee \phi_2 \text{ or } \phi = \phi_1\mathcal{S}\phi_2 \\ \mathsf{Sf}^-(\phi_1) & \text{if } \phi = \mathcal{Y}\phi_1 \text{ or } \phi = \exists x, \phi_1 \\ \emptyset & \text{otherwise,} \end{cases}$$

$$\mathsf{Sf}^+(\phi) \stackrel{def}{=} \{\phi\} \cup \begin{cases} \mathsf{Sf}^-(\phi_1) & \text{if } \phi = \neg\phi_1 \\ \mathsf{Sf}^+(\phi_1) \cup \mathsf{Sf}^+(\phi_2) & \text{if } \phi = \phi_1 \vee \phi_2 \text{ or } \phi = \phi_1\mathcal{S}\phi_2 \\ \mathsf{Sf}^+(\phi_1) & \text{if } \phi = \mathcal{Y}\phi_1 \text{ or } \phi = \exists x, \phi_1 \\ \emptyset & \text{otherwise.} \end{cases}$$

As expected, we define $\mathsf{Sf}(\phi) \stackrel{def}{=} \mathsf{Sf}^-(\phi) \cup \mathsf{Sf}^+(\phi)$.

Example 6. *Consider the following secrecy property*

$$\phi = \neg\exists x, \Diamond(\mathsf{Secret}(a, a, b, x)) \wedge \mathsf{learn}(x)$$

that requires that no term x that has been declared by agent a as secret between agents a and b is known to the attacker. We have the following sets:

$\mathsf{Sf}^+(\phi) = \{\phi\}$

$\mathsf{Sf}^-(\phi) = \{\exists x, \Diamond(\mathsf{Secret}(a, a, b, x)) \wedge \mathsf{learn}(x), \Diamond(\mathsf{Secret}(a, a, b, x)) \wedge \mathsf{learn}(x),$
$\Diamond(\mathsf{Secret}(a, a, b, x)), \mathsf{Secret}(a, a, b, x), \mathsf{learn}(x)\}$

Definition 11 ($\mathcal{PS}$-LTL$^\pm$). $\mathcal{PS}$-LTL$^-$ is the subset of $\mathcal{PS}$-LTL containing only universal quantifiers and in which $\mathsf{learn}(t)$ only occurs negatively, *i.e.*

$$\mathcal{PS}\text{-LTL}^- = \{\phi \in \mathcal{PS}\text{-LTL} \mid \begin{array}{l} \phi = \forall x_1, \ldots \forall x_n, \phi', \\ \phi \text{ closed} \\ \text{no quantifier appears in } \phi' \text{ and} \\ \forall t \in \mathcal{T},\ \mathsf{learn}(t) \notin \mathsf{Sf}^+(\phi') \end{array} \}$$

$\mathcal{PS}$-LTL$^+$ is the subset of $\mathcal{PS}$-LTL containing only existential formulas and in which $\mathsf{learn}(t)$ only occurs positively, *i.e.*

$$\mathcal{PS}\text{-LTL}^+ = \{\phi \in \mathcal{PS}\text{-LTL} \mid \begin{array}{l}\phi = \exists x_1, \dots \exists x_n, \phi', \\ \phi \text{ closed} \\ \text{no quantifier appears in } \phi' \text{ and} \\ \forall t \in \mathcal{T},\ \mathsf{learn}(t) \notin \mathsf{Sf}^-(\phi')\end{array} \}$$

We will often refer to the set of formulas in $\mathcal{PS}$-LTL$^+$as *attack formulas*.

4.2. Security properties

This subsection illustrates the expressivity of $\mathcal{PS}$-LTL, showing in particular how several security properties concerning secrecy and authentication can be expressed using status events and a formula in $\mathcal{PS}$-LTL$^-$.

4.2.1. Secrecy

Secrecy is a security property that characterizes the inability for the intruder to learn a specified term, for example a nonce generated by a participant. More precisely, it states that if an agent has generated a nonce in a session of the protocol with non-compromised agents, then this nonce should never be learned by the intruder. To do so we need to add a status event Secret that is produced just after the creation of this nonce and by the agent who created it. The secrecy property can then be expressed as follows.

$$\phi_{\mathsf{S}} = \forall y, \forall y_1 ... \forall y_k, \forall z, [\Diamond(\mathsf{Secret}(y, y_1, \dots, y_k, z)) \wedge \bigwedge_{i \in [\![k]\!]} \mathsf{NC}(y_i)] \Rightarrow \neg\mathsf{learn}(z)$$

Example 7. *The protocol* Π_{Toy} *presented in Example 3 page 8, to which we add the status event* $\mathsf{Secret}(z_a, z_a, z_b, n)$ *saying that* z_a *has generated a nonce* n *in a session involving* z_a *and* z_b*, does not satisfy secrecy for nonce* n*. The secrecy requirement can be expressed using the formula below.*

$\forall y, \forall y_1 \forall y_2, \forall z, [\Diamond(\mathsf{Secret}(y, y_1, y_2, z)) \wedge \mathsf{NC}(y_1) \wedge \mathsf{NC}(y_2)] \Rightarrow \neg\mathsf{learn}(z)$

The execution obtained applying the substitution of Example 5 page 11 to the symbolic trace of Example 4 page 10 is indeed an attack since: on the one hand n^1 *has been generated in a session only involving honest agents and on the other hand the intruder can deduce* n^1 *from* $\mathsf{aenc}(\langle \mathsf{aenc}(x^3, \epsilon), b\rangle, \epsilon)$ *using decryption with his own key, depairing and decryption again.*

4.2.2. Aliveness

Aliveness is the weakest authentication property in Lowe's hierarchy [Low97]. It states that whenever an agent a finishes a non-compromised session, allegedly with another agent b, then b must have once started a session of the same protocol. This definition is quite weak since there is no additional condition on the role b played or who else was involved in this session. In order to express this property, for a protocol involving $z_1, ..., z_k$, we need to insert two status events per role.

First, at the beginning of the description of the protocol, we add the status events $\mathsf{Start}_i(z_i)$ (for all $i \in [\![k]\!]$) then, at the end of the protocol, we add the status events $\mathsf{End}_i(z_i, z_1, ..., z_k)$ (for all $i \in [\![k]\!]$) meaning that z_i has finished executing the protocol with participants $z_1, ..., z_k$. The aliveness property can then be expressed as follows.

$$\begin{array}{rcl} \phi_{\mathsf{A}} & = & \forall y, \forall y_1, \ldots \forall y_k, \\ & & \bigwedge\limits_{j \in [\![k]\!]} [(\mathsf{End}_j(y, y_1, ..., y_k) \wedge \bigwedge\limits_{i \in [\![k]\!]} \mathsf{NC}(y_i)) \Rightarrow \bigwedge\limits_{\substack{i \in [\![k]\!] \\ i \neq j}} \bigvee\limits_{h \in [\![k]\!]} \Diamond(\mathsf{Start}_h(y_i))] \end{array}$$

Example 8. *Adding the above mentioned status events to* Π_{Toy} *gives the protocol:*

$$\Pi_{\mathsf{Toy}} = \left[\begin{array}{l} \mathsf{Start}_1(z_a); \mathsf{Start}_2(z_b); \\ \mathsf{snd}(z_a, z_b, \mathsf{aenc}(\langle \mathsf{aenc}(n, z_b), z_a \rangle, z_b)); \\ \mathsf{rcv}(z_b, z_a, \mathsf{aenc}(\langle \mathsf{aenc}(x, z_b), z_a \rangle, z_b)); \\ \mathsf{snd}(z_b, z_a, \mathsf{aenc}(\langle \mathsf{aenc}(x, z_a), z_b \rangle, z_a)); \\ \mathsf{rcv}(z_a, z_b, \mathsf{aenc}(\langle \mathsf{aenc}(n, z_a), z_b \rangle, z_a)); \\ \mathsf{End}_1(z_a, z_a, z_b); \mathsf{End}_2(z_b, z_a, z_b) \end{array}\right]$$

for which the aliveness can be expressed by the formula

$$\begin{array}{rl} & \forall y, \forall y_1, \forall y_2, \\ & (\mathsf{End}_1(y, y_1, y_2) \wedge \mathsf{NC}(y_1) \wedge \mathsf{NC}(y_2)) \Rightarrow (\Diamond(\mathsf{Start}_1(y_2)) \vee \Diamond(\mathsf{Start}_2(y_2)) \\ \wedge & (\mathsf{End}_2(y, y_1, y_2) \wedge \mathsf{NC}(y_1) \wedge \mathsf{NC}(y_2)) \Rightarrow (\Diamond(\mathsf{Start}_1(y_1)) \vee \Diamond(\mathsf{Start}_2(y_1)) \end{array}$$

Considering the scenario $\mathsf{sc} = [(z_b, 1); (z_b, 1); (z_b, 1); (z_b, 1)]$ *and a function* α *defined by* $\alpha(1, z_a) = a$ *and* $\alpha(1, z_b) = b$ *we obtain the following symbolic trace.*

$$\mathsf{Start}_2(b); \mathsf{rcv}(b, a, \mathsf{aenc}(\langle \mathsf{aenc}(x^1, b), a \rangle, b)); \mathsf{snd}(b, a, \mathsf{aenc}(\langle \mathsf{aenc}(x^1, a), b \rangle, a)); \mathsf{End}_2(b, a, b)$$

Considering the substitution $\sigma = \{x^1 \mapsto \epsilon\}$ *we can see that the intruder can generate the only message received by* b *using pairing and encryption with* b*'s public key. Thus* b *can run a session to its end, thinking that he is communicating with* a *but without* a *ever running any session of the protocol, which consequently breaks aliveness.*

4.2.3. Weak agreement

Weak agreement is an authentication property slightly stronger than aliveness. It states that whenever an agent a finishes a non-compromised session allegedly with another agent b then b must have once started a session of the same protocol in which a was also believed to take part. Again there is no additional condition on the role b played in this session. In order to express this property, for a protocol involving $z_1, ..., z_k$, we need to insert new status events. First at the beginning of the description of the protocol we insert the status events $\mathsf{Start}_{ij}(z_i, z_j)$ for every couple of roles with $i \neq j$, stating that z_i playing role i has started a session, allegedly communicating (among others)

with z_j playing role j. Then at the end of the protocol we insert one status event $\mathsf{End}_i(z_i, z_1, ..., z_k)$ per role, meaning that z_i playing role i has finished executing the protocol with participants $(z_1, ..., z_k)$. With these status events the following formula expresses weak agreement.

$$\phi_{\mathsf{WA}} = \forall y, \forall y_1, \ldots \forall y_k,$$
$$\bigwedge_{i \in [\![k]\!]} [(\mathsf{End}_i(y, y_1, ..., y_k) \wedge \bigwedge_{j \in [\![k]\!]} \mathsf{NC}(y_j)) \Rightarrow \bigwedge_{\substack{j \in [\![k]\!] \\ j \neq i}} \bigvee_{h_1 \in [\![k]\!]} \bigvee_{h_2 \in [\![k]\!]} \Diamond(\mathsf{Start}_{h_1 h_2}(y_j, y_i))]$$

Example 9. *Adding the above mentioned status events to* Π_{Toy} *gives the following protocol.*

$$\Pi_{\mathsf{Toy}} = \left[\begin{array}{l} \mathsf{Start}_{12}(z_a, z_b); \mathsf{Start}_{21}(z_b, z_a); \\ \mathsf{snd}(z_a, z_b, \mathsf{aenc}(\langle \mathsf{aenc}(n, z_b), z_a \rangle, z_b)); \\ \mathsf{rcv}(z_b, z_a, \mathsf{aenc}(\langle \mathsf{aenc}(x, z_b), z_a \rangle, z_b)); \\ \mathsf{snd}(z_b, z_a, \mathsf{aenc}(\langle \mathsf{aenc}(x, z_a), z_b \rangle, z_a)); \\ \mathsf{rcv}(z_a, z_b, \mathsf{aenc}(\langle \mathsf{aenc}(n, z_a), z_b \rangle, z_a)); \\ \mathsf{End}_1(z_a, z_a, z_b); \mathsf{End}_2(z_b, z_a, z_b) \end{array}\right]$$

for which the weak agreement can be expressed by the formula

$$\forall y, \forall y_1, \forall y_2,$$
$$(\mathsf{End}_1(y, y_1, y_2) \wedge \mathsf{NC}(y_1) \wedge \mathsf{NC}(y_2)) \Rightarrow (\Diamond(\mathsf{Start}_{12}(y_2, y_1)) \vee \Diamond(\mathsf{Start}_{21}(y_2, y_1)))$$
$$\wedge (\mathsf{End}_2(y, y_1, y_2) \wedge \mathsf{NC}(y_1) \wedge \mathsf{NC}(y_2)) \Rightarrow (\Diamond(\mathsf{Start}_{12}(y_1, y_2)) \vee \Diamond(\mathsf{Start}_{21}(y_1, y_2)))$$

Since our protocol does not satisfy the (weaker) aliveness property, it does not satisfy weak agreement either.

4.2.4. Non-injective agreement

The last security property we will consider in detail in this paper is a restricted version of Lowe's non-injective agreement. In addition to the weak agreement property described above, it gives information about the roles the two agents played in their respective sessions and the data exhanged during those. It states that whenever an agent a finishes a non-compromised session, playing role i allegedly with another agent b playing role j, and thinking that they both have agreed on some value t, then b must have once started a session of the same protocol in which he played role j, in which a was playing role i, and also thinks they have agreed on this same value t.

In order to express this property, we reuse the status events introduced for weak agreement. For simplicity we omit the agreement on the data t, and only focus on the agreement on the participants to the sessions. But it is easy to also include agreement on data values different from agent names. The non-injective agreement can then be expressed using the following formula.

$$\phi_{\mathsf{NIA}} = \forall y, \forall y_1, \ldots \forall y_k,$$
$$\bigwedge_{i \in [\![k]\!]} [(\mathsf{End}_i(y, y_1, ..., y_k) \wedge \bigwedge_{j \in [\![k]\!]} \mathsf{NC}(y_j)) \Rightarrow \bigwedge_{\substack{j \in [\![k]\!] \\ i \neq j}} \Diamond(\mathsf{Start}_{ji}(y_j, y_i))]$$

Example 10. *We can reuse the protocol of Example 9 for which the weak agreement can be expressed by the formula below.*

$$\begin{aligned} \forall y, \forall y_1, \forall y_2, \ & [(\mathsf{End}_1(y, y_1, y_2) \wedge \mathsf{NC}(y_1) \wedge \mathsf{NC}(y_2)) \Rightarrow (\Diamond(\mathsf{Start}_{21}(y_2, y_1)))] \\ & \wedge [(\mathsf{End}_2(y, y_1, y_2) \wedge \mathsf{NC}(y_1) \wedge \mathsf{NC}(y_2)) \Rightarrow (\Diamond(\mathsf{Start}_{12}(y_1, y_2)))] \end{aligned}$$

Since our protocol does not satisfy the (weaker) aliveness property, it does not satisfy weak agreement either.

Using formulas of $\mathcal{PS}$-LTL$^-$ it is also possible to capture properties like *fairness* which is relevant in electronic contract signing protocols, or as *temporary secrecy* which is important in electronic voting protocols. We will not detail here the formalisation of these properties. We refer the interested reader to [CDD07] for formal definitions of *fairness* and *temporary secrecy* in $\mathcal{PS}$-LTL$^-$.

5. Well-formed protocols and well-typed attacks

In this section, we present the main result of the paper. It states that for *well-formed* protocols (*i.e.* with non-unifiable subterms), for the verification of a class of interesting security properties, including those described in section 4.2, we only need to consider well-typed runs of the protocol when looking for an attack. In other words, we exhibit a class of protocols and a class of properties for which the typing abstraction (with respect to the type system that will be introduced in section 5.2) is correct.

The section starts with an informal statement of the results, followed by the introduction of the strong typing system we use to restrict the search space for attacks. We can then present more formally our main theorem (Theorem 1 page 27). The formal proof of this result is given in Section 6.3.

5.1. What do we prove?

Before introducing the formal definition of our typing system and stating precisely the main result of this paper, we first try to present all these notions informally.

The strength of Theorem 1 lies in the fact that, under certain conditions on the protocol and the property, we significantly reduce the search space for attacks. To do so we introduce a very strong typing system.

Typing system. The particularity of our typing system is that a different type is introduced for every nonce and every constant appearing in the protocol. The interest of this distinctness is that in the end we will prove that there is no need to consider attacks that are not well-typed, for example attacks in which a nonce is reused where it is not supposed to be, or in which a more complex message is used instead of a simple nonce. Hence the stronger the typing system, the more powerful the result. Our type system is a refinement of the one introduced in [HLS03].

A protocol is said to be *well-typed* whenever any given status event is always used with parameters of the same types. In other words for two occurrences of the same status event in a protocol, the types of the parameters in both occurrences are coherent (see Definition 15 page 21). A formula is said to be *well-typed* when it is in coherence with the types induced by the protocol. More precisely it requires that a variable from the formula is given exactly one type, that the corrupted function C is only applied to agent type terms and that whenever an agent name appears in the formula as the parameter of a status event, this parameter has type agent in the protocol.

Class of protocols. Our result does not apply to all protocols but to a reasonable subset. We require that:

- the protocol is well-typed
- asymmetric encryption is used coherently, *i.e.* whenever the subterm $\mathsf{aenc}(u, v)$ occurs in Π, then v is of type agent, denoted α
- Two encrypted subterms of a symbolic trace of the protocol that don't have the same type are not unifiable.

The third condition seems to be the most restrictive one. In particular our toy example does not satisfy it (see Example 17 page 24). However, this restriction is not as drastic as it may appear. Indeed, we prove in Section 7.1 that, even for protocols that would not satisfy this condition, a light tagging scheme would, without limiting the possible honest executions of the protocol, make it fit in our characterization.

We will illustrate the notions presented in this section on a simple protocol. Since we want to use the simple and informal notation of Example 3 page 8, we need to introduce the notation $a : Q(a, t_1, ..., t_n)$ for status events introduced in Section 3.1 of the paper.

Example 11. *Let us now consider the following protocol* Π_{ex}*.*

$$\begin{array}{lcll} a & \to & b: & \{1, a, N_a, N'_a\}_b \\ b & & : & \mathsf{Secret}(b, a, b, N_b) \\ b & \to & a: & \{2, N_b, N_a\}_a \\ a & \to & b: & \{3, N_b, N_a\}_b \end{array}$$

In this protocol agent a generates two nonces, adds a constant 1 and his identity and sends it encrypted with b's public key. Agent b generates a nonce,

that is supposed to remain secret, and sends it back to a together with the first nonce a sent and a new constant 2, all encrypted with a's public key. Agent a then decrypts the received message and sends back both nonces together with the constant 3 back to b and encrypted with b's public key.

Now since there is just one occurrence of a status event, the protocol is obviously well-typed. Furthermore asymmetric encryption is used coherently (only agent names are used as public keys). For the last condition, each encrypted subterm generated in the protocol has to contain a different constant which, as we will see in Section 7.1, enforces the non unifiability of subterms of different types as desired. This protocol thus belongs to our class.

Class of properties. The class of properties to which our result applies is defined with respect to a given protocol. It requires several conditions, among which the well-typedness of the formula. It is worth noting in particular that the properties of secrecy and authentication introduced in Section 4.2 belong to this class for any reasonable protocol of the above class, reasonable meaning roughly speaking that all status events necessary for the property are in the protocol and have been introduced as suggested in Section 4.2.

Example 12. *On our example protocol Π_{ex}, we want to ensure that if b starts a session, allegedly with a non corrupted agent, then the nonce N_b is never learned by the intruder. This can be expressed as follows.*

$\Phi_{ex} = \forall y, \forall y_1, \forall y_2, \forall z, [\Diamond(\mathsf{Secret}(y, y_1, y_2, z)) \wedge \mathsf{NC}(y_1) \wedge \mathsf{NC}(y_2)] \Rightarrow \neg\mathsf{learn}(z)$

Our result. Using the previously defined classes of protocols and properties, Theorem 1 can be informally stated as follows.

Given a protocol Π and a property ϕ belonging to the above classes, then if the protocol admits an attack on ϕ (*i.e.* a valid execution that does not satisfy ϕ) the protocol also admits a well-typed attack on ϕ, *i.e.* an attack in which each subterm of each message has the type decided by the original protocol.

As said before, this significantly reduces the search space when looking for attacks as we just need to check well-typed executions. In particular, this result implies that the size of messages to consider is bounded by the maximal size of a message in the protocol, as established in Corollary 1 page 47.

Example 13. *As illustrated in the two previous examples, protocol Π_{ex} and formula Φ_{ex} belong to our classes of protocol and formula respectively. Our result then applies and ensures that if an attack exists, a well-typed one also exists. This does not mean that only well-typed attacks exist, as we will illustrate now. The following informal trace corresponds to a badly typed attack. In this attack, a is willing to communicate with the intruder i, but b believes he is talking to a. The nonce generated by b thus should remain secret from the intruder, which is not the case. To take into account the fact that the intruder can steal/forge messages, in the following executions all messages created by honest agents are sent to the intruder and only he can send messages to those agents, possibly pretending to be someone else (notation $\epsilon(x)$). Note that at each step the intruder can create all the received messages.*

$$
\begin{array}{lll}
a & \to \quad \epsilon : & \{1, a, N_a, N'_a\}_\epsilon \\
\epsilon(a) & \to \quad b : & \{1, a, N_a, \langle N'_a, N'_a\rangle\}_b \\
b & \quad : & \mathsf{Secret}(b, a, b, N_b) \\
b & \to \epsilon(a) : & \{2, N_b, N_a\}_a \\
\epsilon & \to \quad a : & \{2, N_b, N_a\}_a \\
a & \to \quad \epsilon : & \{3, N_b, N_a\}_\epsilon \\
\epsilon(a) & \to \quad b : & \{3, N_b, N_a\}_b
\end{array}
$$

In the previous attack the intruder uses the fact that a wants to initiate a session with him to start a session with b, pretending to be a, and fools b in doing so. He decrypts a's message to himself and encrypts it for b, pretending to be a. Agent b doesn't notice that the message comes from the intruder and follows the protocol, appending his nonce and sending the message, allegedly to a. The intruder then uses a to decrypt this message and get b's nonce that he can then resend. It is an attack since the secret of b is supposed to be known only by a and b and ϵ manages to get it. The attack is not well-typed since b receives a pair $\langle N'_a, N'_a\rangle$ when he was just expecting a nonce. Note that since b expects a nonce, hence an unidentified data, he cannot notice that the received data is a pair and not a nonce.

The previous badly-typed attack has a corresponding well-typed version, that is given below.

$$
\begin{array}{lll}
a & \to \quad \epsilon : & \{1, a, N_a, N'_a\}_\epsilon \\
\epsilon(a) & \to \quad b : & \{1, a, N_a, N'_a\}_b \\
b & \quad : & \mathsf{Secret}(b, a, b, N_b) \\
b & \to \epsilon(a) : & \{2, N_b, N_a\}_a \\
\epsilon & \to \quad a : & \{2, N_b, N_a\}_a \\
a & \to \quad \epsilon : & \{3, N_b, N_a\}_\epsilon \\
\epsilon(a) & \to \quad b : & \{3, N_b, N_a\}_b
\end{array}
$$

This example shows that a protocol in our class can still have an attack, and even a badly-typed one. Our result claims that, even though such badly-typed executions may exist, there is no need to consider them to prove that the protocol satisfies or not a property.

5.2. Types

The goal of this section is to define the typing relation, denoted $\mathcal{E} \triangleright F : \tau$, which in the *typing environment* $\mathcal{E}$ associates the type τ to the term or predicate F. The notion of *well-typed execution*, is defined on top of this typing relation.

Definition 12 (Typing environment). A typing environment $\mathcal{E}$ is a pair $\langle \Pi, \phi\rangle$ with:

- a protocol Π, and
- a formula $\phi \in \mathcal{PS}\text{-LTL}^-$.

We define the set of names and variables of $\mathcal{E}$ as follows

$$\begin{array}{lcl} \mathcal{N}(\mathcal{E}) & = & \{n, n^{sid}, n^{\epsilon,k} \mid n \in \mathcal{N}(\Pi) \ \wedge \ n^{sid} \in \mathcal{N} \ \wedge \ n^{\epsilon,k} \in \mathcal{N}^{\epsilon} \ \wedge \ sid, k \in \mathbb{N}\} \\ \mathcal{X}(\mathcal{E}) & = & \{x, x^{sid} \mid x \in \mathcal{X}(\Pi), \ sid \in \mathbb{N}\} \ \cup \ \mathcal{X}(\phi) \end{array}$$

Before defining the typing relation $\triangleright$, we need to determine the set of considered types. This set is built from the specification of the protocol Π.

Definition 13 (Type induced by Π). For each nonce $n \in \mathcal{N}(\Pi)$ we introduce a new type ν_n which will be associated to it. For each constant $c \in \mathcal{C}$ we introduce a new type γ_c which will be associated to it. We also introduce the type α which will be associated to agents, and the undefined type ω. The set of types induced by Π is defined by the following grammar.

$$\begin{array}{lcll} \tau, \tau_1, \ldots & ::= & \alpha & \\ & | & \nu_n & n \in \mathcal{N}(\Pi) \\ & | & \gamma_c & c \in \mathcal{C} \\ & | & \omega & \\ & | & \mathsf{pv}(\alpha) & \\ & | & \mathsf{sh}(\alpha, \alpha) & \\ & | & f(\tau_1, \ldots, \tau_n) & f \in \{\langle\rangle, \mathsf{senc}, \mathsf{aenc}, \mathsf{sign}, \mathsf{h}\} \end{array}$$

The undefined type ω is introduced to be associated to objects to which Π doesn't give a type, in other words status events, nonces and variables not appearing in Π.

Definition 14 (Typing relation $\mathcal{E} \triangleright F : \tau$). Let Π be a protocol, ϕ be a formula, τ be a type induced by Π, and F be a term or a predicate. We will say that F is of type τ in the environment $\mathcal{E} = \langle \Pi, \phi \rangle$, denoted $\mathcal{E} \triangleright F : \tau$, if there exists a tree whose nodes are labelled with expressions of the form $\mathcal{E} \triangleright F' : \tau'$ and such that:

- the root is labelled with $\mathcal{E} \triangleright F : \tau$,
- the leaves have an empty label,
- for every node labelled $\mathcal{E} \triangleright F' : \tau'$ with sons labelled $\mathcal{E} \triangleright F'_1 : \tau'_1 \ \ldots \ \mathcal{E} \triangleright F'_n : \tau'_n$, we have that $\dfrac{\mathcal{E} \triangleright F'_1 : \tau'_1 \ \ldots \ \mathcal{E} \triangleright F'_n : \tau'_n}{\mathcal{E} \triangleright F' : \tau'}$ is an instance of one of the inference rules given in Figure 2.

In Definition 13 we introduced the type α for typing agent names and variables. This is precisely what rule (1) describes. Similarly, for all $i \in [\![m]\!]$ we introduced the type γ_i for the constant c_i and rule (6) precisely assigns γ_i to c_i. In Definition 13 we also introduced a different type ν_i for each nonce n_i appearing in the specification of Π with the purpose of typing each instance n_i^{sid} with type ν_i. This is precisely what rules (3) and (5) do. Furtheremore, we give the intruder as many nonces of type ν_i as he needs (rule (4)). It is important to note that in this way our type system (rules (3)-(6)) associates a different type to each nonce and each constant occurring in Π. To nonces and variables not

$$\frac{}{\mathcal{E} \rhd t : \alpha}\ t \in \mathcal{A} \cup \mathcal{Z} \quad (1) \qquad \frac{}{\mathcal{E} \rhd t : \omega}\ t \in (\mathcal{N} \cup \mathcal{X}) \setminus (\mathcal{N}(\mathcal{E}) \cup \mathcal{X}(\mathcal{E})) \quad (2)$$

$$\frac{}{\mathcal{E} \rhd n : \nu_n}\ n \in \mathcal{N}(\Pi) \quad (3) \qquad \frac{}{\mathcal{E} \rhd n^{\epsilon,k} : \nu_n}\ n \in \mathcal{N}(\Pi) \quad (4) \qquad \frac{}{\mathcal{E} \rhd n^{sid} : \nu_n}\ n \in \mathcal{N}(\Pi) \quad (5)$$

$$\frac{}{\mathcal{E} \rhd c : \gamma_c}\ c \in \mathcal{C} \quad (6) \qquad \frac{\mathcal{E} \rhd t_1 : \tau_1 \ \ldots \ \mathcal{E} \rhd t_n : \tau_n}{\mathcal{E} \rhd f(t_1, \ldots, t_n) : f(\tau_1, \ldots, \tau_n)}\ f \in \mathcal{F} \quad (7)$$

$$\frac{\mathcal{E} \rhd \delta_\Pi(x) : \tau}{\mathcal{E} \rhd x : \tau}\ x \in \mathcal{X}(\Pi) \quad (8) \qquad \frac{\mathcal{E} \rhd \delta_\Pi(x) : \tau}{\mathcal{E} \rhd x^{sid} : \tau}\ x \in \mathcal{X}(\Pi) \quad (9)$$

$$\frac{\mathcal{E} \rhd t_1 : \tau_1 \ \ldots \mathcal{E} \rhd t_n : \tau_n}{\mathcal{E} \rhd \mathsf{Q} : \tau_1 \times \cdots \times \tau_n}\ \mathsf{Q}(t_1, \ldots, t_n) \in \mathsf{Evts}(\Pi) \quad (10) \qquad \frac{}{\mathcal{E} \rhd \mathsf{Q} : \omega}\ \begin{array}{l}\forall t_1, \ldots, t_n, \\ \mathsf{Q}(t_1, \ldots, t_n) \notin \mathsf{Evts}(\Pi)\end{array} \quad (11)$$

$$\frac{\mathcal{E} \rhd \mathsf{Q} : \tau_1 \times \cdots \times \tau_n}{\mathcal{E} \rhd u_i : \tau_i}\ \begin{array}{l} i \in [\![n]\!] \\ \mathsf{Q}(u_1, \ldots, u_n) \in \mathsf{Sf}(\phi) \\ u_i \in \mathcal{X}\end{array} \quad (12)$$

Figure 2: Typing rules

appearing in the specification of Π we just assign the undefined type ω (rule (2)). Variables of a symbolic trace are assigned with their "expected type", that is the type that honest agents expect for the terms that will be instantiating them (rules (8) and (9)). Finally, the types of variables occuring in a formula are derived from the types prescribed by the protocol specification (rules (10)-(12)). Indeed, the protocol specification imposes a type on each status event Q (rules (10) and (11)), which imposes the type of the variables appearing under Q in the considered formula (rule (12)). Rule (7) prescribes how the type of a complex term is derived from the types of its direct subterms in the expected way.

Definition 15 (Well-typed protocol). We say that a protocol $\Pi = [\mathsf{e}_1; \ldots; \mathsf{e}_\ell]$ is well-typed in the typing environment $\mathcal{E}$ whenever $\forall i, j \in [\![\ell]\!]$, if we have $e_i = \mathsf{Q}(p, u_1, \ldots, u_q)$ and $e_j = \mathsf{Q}(p', v_1, \ldots, v_r)$, then $q = r$ and for all $k \in [\![q]\!]$ there exists a unique type $\tau_k \neq \omega$ such that $\mathcal{E} \rhd u_k : \tau_k$ and $\mathcal{E} \rhd v_k : \tau_k$.

In other words, all occurrences of Q in Π have the same type in the above type system. Concerning formulas, the following definition characterises the formulas which are in accordance with the types induced by Π.

Example 14. *The three versions of protocol Π_{Toy} introduced in Examples 7 to 9 pages 13 to 15 are all well-typed since each status event appears at most once in the protocol.*

Definition 16 (Well-typed formula). We say that $\phi \in \mathcal{PS}\text{-LTL}^{-}$ is well-typed in the environment $\mathcal{E}$ if

- for all $x \in \mathcal{X}(\phi)$, there exists a unique type $\tau \neq \omega$ such that $\mathcal{E} \triangleright x : \tau$,
- for all $\mathsf{C}(u) \in \mathsf{Sf}(\phi)$, $\mathcal{E} \triangleright u : \alpha$,
- for all $\mathsf{Q}(t_1, \ldots, t_n) \in \mathsf{Sf}(\phi)$ with $\mathcal{E} \triangleright \mathsf{Q} : \tau_1 \times \cdots \times \tau_n$, for all $i \in [\![n]\!]$, if $t_i \in \mathcal{A}$, then $\tau_i = \alpha$.

The above definition requires that all variables appearing in the formula are given exactly one valid type (not undefined), that the corrupted function C is only applied to terms of agent type and that if an agent name appears as a parameter of a status event in the formula, then this parameter has type agent in the protocol.

Example 15. *All formulas of Section 4.2 are, if used with a reasonable protocol, well-typed. As an example, let us consider the protocol and formula of Example 8 page 14.*

$$\begin{array}{rl} & \forall y, \forall y_1, \forall y_2, \\ & (\mathsf{End}_1(y, y_1, y_2) \wedge \mathsf{NC}(y_1) \wedge \mathsf{NC}(y_2)) \Rightarrow (\Diamond(\mathsf{Start}_1(y_2)) \vee \Diamond(\mathsf{Start}_2(y_2)) \\ \wedge & (\mathsf{End}_2(y, y_1, y_2) \wedge \mathsf{NC}(y_1) \wedge \mathsf{NC}(y_2)) \Rightarrow (\Diamond(\mathsf{Start}_1(y_1)) \vee \Diamond(\mathsf{Start}_2(y_1))) \end{array}$$

All variables are used first in status event End_1 which gives them type α, which in turn ensures all three conditions of well-typedness.

The following lemma establishes that our type system is coherent, and in particular that we have indeed defined a typing relation, *i.e.* given a term t, the typing relation $\triangleright$ in the well-typed environment $\mathcal{E}$ attribues one and only one type to t.

Lemma 1. *Let Π be a protocol and ϕ a formula such that Π and ϕ are well-typed in the typing environment $\mathcal{E} = \langle \Pi, \phi \rangle$. Given a term t, there is exactly one type τ such that $\mathcal{E} \triangleright t : \tau$.*

The proof of this lemma is rather simple and done by induction on t, using the typing rules of Figure 2 page 21.

Definition 17 ($\mathcal{E} \triangleright \sigma$). Let Π be a protocol and ϕ a formula such that Π and ϕ are well-typed in the typing environment $\mathcal{E} = \langle \Pi, \phi \rangle$. We will say that a substitution $\sigma = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ is well-typed in the environment $\mathcal{E}$, denoted $\mathcal{E} \triangleright \sigma$, if for all $i \in [\![n]\!]$ and for all types τ_i

$$\mathcal{E} \triangleright x_i : \tau_i \;\Leftrightarrow\; \mathcal{E} \triangleright t_i : \tau_i$$

Lemma 2. *Let Π be a protocol and ϕ a formula such that Π and ϕ are well-typed in the typing environment $\mathcal{E} = \langle \Pi, \phi \rangle$. Given a term t, a type τ and a substitution σ. If $\mathcal{E} \triangleright t : \tau$ and $\mathcal{E} \triangleright \sigma$, then $\mathcal{E} \triangleright t\sigma : \tau$.*

The proof of this lemma is rather simple and done again by induction on t, using the typing rules of Figure 2 page 21.

5.3. Considering only well-typed attacks

Before presenting the main result of the paper and the intermediate propositions used to prove this theorem, we introduce the classes of protocols $\mathbb{P}$ and of properties $\Phi(\Pi)$ to which our result applies. Our choice is to give for these classes a characterization as simple as possible, to make them easy to understand and to check on concrete protocols/formulas. We thus need to present and prove Proposition 1 page 24 in order to have an equivalent characterization that will be easier to use in the proofs to come.

The definition of our class of protocols relies on a particular trace called witness trace including one complete session per role, and involving only one agent playing all roles.

Definition 18 (Witness trace $\mathsf{tr}_{\mathtt{wtn}}(\Pi)$). Let Π be a k-party protocol with $\mathsf{Partcpts}(\Pi) = \{r_1, \ldots, r_k\}$, and $\mathsf{length}(\Pi(r_j)) = \ell_j$. Let $\mathsf{tr}_{\mathtt{wtn}}(\Pi)$ be the symbolic trace associated to the following scenario

$$\mathsf{sc}_{\mathtt{wtn}} = [\underbrace{(r_1,1);\ldots;(r_1,1)}_{\ell_1 \text{ times}};\ldots;\underbrace{(r_k,k);\ldots;(r_k,k)}_{\ell_k \text{ times}}]$$

and to the following instantiation function

$$\forall i \in [\![k]\!],\ \forall z \in \mathcal{Z}(\Pi),\ \alpha(z,i) = a$$

for some $a \in \mathcal{A}$

Example 16. *The witness trace $\mathsf{tr}^{\mathsf{Toy}}_{\mathtt{wtn}}$ for our running example Π_{Toy} corresponds to the interleaving $intl^{\mathsf{Toy}}_{\mathtt{wtn}} = [(z_a,1);(z_a,1);(z_b,2);(z_b,2)]$ and the instantiation function $\alpha^{\mathsf{Toy}}_{\mathtt{wtn}}$ defined by $\alpha^{\mathsf{Toy}}_{\mathtt{wtn}}(z_a,1) = \alpha^{\mathsf{Toy}}_{\mathtt{wtn}}(z_b,1) = \alpha^{\mathsf{Toy}}_{\mathtt{wtn}}(z_a,2) = \alpha^{\mathsf{Toy}}_{\mathtt{wtn}}(z_b,2) = a$, and consists of the following four events:*

$$\mathsf{tr}^{\mathsf{Toy}} = \left[\begin{array}{l} \mathsf{snd}(a,a,\mathsf{aenc}(\langle \mathsf{aenc}(n^1,a),a\rangle,a));\\ \mathsf{rcv}(a,a,\mathsf{aenc}(\langle \mathsf{aenc}(n^1,a),a\rangle,a));\\ \mathsf{rcv}(a,a,\mathsf{aenc}(\langle \mathsf{aenc}(x^2,a),a\rangle,a));\\ \mathsf{snd}(a,a,\mathsf{aenc}(\langle \mathsf{aenc}(x^2,a),a\rangle,a)) \end{array}\right]$$

Definition 19 (The class of protocols $\mathbb{P}$). The class of protocols $\mathbb{P}$ is the set of protocols Π satisfying the three following conditions:

1. Π is well-typed in the environment $\mathcal{E} = \langle \Pi, \mathsf{true} \rangle$;
2. $\forall \mathsf{aenc}(u, v) \in \mathsf{St}(\Pi),\ \langle \Pi, \mathsf{true} \rangle \triangleright v : \alpha$; and
3. $\forall u, v \in \mathsf{CryptSt}(\mathsf{tr}_{\mathtt{wtn}}(\Pi)),\ \mathrm{mgu}(u, v) \neq \bot \ \Rightarrow$
$\exists \tau,\ \langle \Pi, \mathsf{true} \rangle \triangleright u : \tau \ \wedge\ \langle \Pi, \mathsf{true} \rangle \triangleright v : \tau$

Item 2 of Definition 19 only constrains the type for the second component of asymmetric encryption. The distinction is due to the fact that, for asymmetric encryption, the decryption of $\mathsf{aenc}(u, v)$ requires to know the private key of v and, for typing reasons, this v has to be of type agent. Symmetric encryption and signing do not have such a constraint as signing does not require any decryption key, and for symmetric encryption there is no constraint on the type of the key.

Example 17. *The witness trace* $\mathsf{tr}^{\mathsf{Toy}}_{\mathtt{wtn}}$ *for our running example (detailed at Example 16) does not satisfy Condition 3 of Definition 19. Indeed, we have that* $\mathsf{aenc}(x^2, a)$ *and* $\mathsf{aenc}(\langle \mathsf{aenc}(n^1, a), a \rangle, a))$ *are unifiable subterms of* $\mathsf{CryptSt}(\mathsf{tr}^{\mathsf{Toy}}_{\mathtt{wtn}})$ *but of different type. According to our type system*

$$\langle \Pi, \mathsf{true} \rangle \triangleright \mathsf{aenc}(x^2, a) : \mathsf{aenc}(\nu_n, \alpha)$$

while

$$\langle \Pi, \mathsf{true} \rangle \triangleright \mathsf{aenc}(\langle \mathsf{aenc}(n^1, a), a \rangle, a)) : \mathsf{aenc}(\langle \mathsf{aenc}(\nu_n, \alpha), \alpha \rangle, \alpha))$$

Thus Π_{Toy} *is not a protocol in the class* $\mathbb{P}$*. However, all four versions, the original one without status events and the others introduced in Examples 7 to 9 pages 13 to 15, are well-typed (see Example 14 page 22) and satisfy condition 2 of the above definition since the only terms appearing as a second parameter of asymmetric encryption are agent variables* z_a *and* z_b*. In Section 7.1 we discuss how cryptographic protocols satisfying the conditions of Definition 4 page 7 can be transformed into protocols in* $\mathbb{P}$ *using static tags. In particular, we show at Example 22 page 49 how to transform* Π_{Toy} *into an "equivalent" protocol in* $\mathbb{P}$*.*

Proposition 1. *Let* $\Pi = [\mathsf{e}_1; \ldots; \mathsf{e}_\ell]$ *be a protocol.* Π *is in* $\mathbb{P}$ *if and only if it satisfies the three following conditions:*

1'. For every formula $\phi \in \mathcal{PS}\text{-}LTL^-$*,* Π *is well-typed in the environment* $\mathcal{E} = \langle \Pi, \phi \rangle$*;*
2'. For every formula $\phi \in \mathcal{PS}\text{-}LTL^-$*,* $\forall \mathsf{aenc}(u, v) \in \mathsf{St}(\Pi),\ \langle \Pi, \phi \rangle \triangleright v : \alpha$*; and*
3'. For every formula $\phi \in \mathcal{PS}\text{-}LTL^-$ *and every symbolic trace* tr *of* Π*,* $\forall u, v \in \mathsf{CryptSt}(\mathsf{tr}),\ \mathrm{mgu}(u, v) \neq \bot \ \Rightarrow\ \exists \tau,\ \langle \Pi, \phi \rangle \triangleright u : \tau \ \wedge\ \langle \Pi, \phi \rangle \triangleright v : \tau$*.*

Proof. One implication is obvious. We just need to prove *1* $\Rightarrow$ *1'*, *2* $\Rightarrow$ *2'*, and *3* $\Rightarrow$ *3'*. But before proceeding with the proof of these implications, we will first establish that for every pair of formulas $\phi_1, \phi_2 \in \mathcal{PS}\text{-LTL}^-$, every term $t \in \mathsf{St}(\Pi) \cup \mathsf{St}(\delta_\Pi(\Pi))$ and every type τ,

$\mathcal{E}_1 \triangleright t : \tau$ if and only if $\mathcal{E}_2 \triangleright t : \tau$ (†)

where $\mathcal{E}_i = \langle \Pi, \phi_i \rangle$ for $i \in \{1, 2\}$. It is enough to prove one of the two implications, since they are symmetric. We proceed by induction on the depth of $\langle \Pi, \phi_1 \rangle \triangleright t : \tau$.

Case $t \in \mathcal{A} \cup \mathcal{Z}$ The only rule that can be applied does not depend on the formula in the considered typing environment (rule (1) from Figure 2 page 21). Hence in both typing environments t is of type α.

Case $t \in \mathcal{N}$**.** As above the only rule from Figure 2 that can be applied (rule (3)) does not depend on the formula and t has the same type in both typing environments. Indeed, because by hypothesis $t \in \mathsf{St}(\Pi) \cup \mathsf{St}(\delta_\Pi(\Pi))$, and $t \in \mathcal{N} \cup \mathcal{C}$, it must be that $t \in \mathcal{N}(\Pi) \subseteq \mathcal{N}(\mathcal{E}_i)$. Thus, rule (2) is not applicable, and the only applicable rule (rule (3)) does not depend on the environment $\mathcal{E}_i$.

Case $t \in \mathcal{C}$**.** As above the only rule from Figure 2 that can be applied is rule (6) and does not depend on the formula and t has the same type in both typing environments.

Case $t \in \mathcal{X}$**.** In that case, the only rule that can be applied is rule (8). Indeed, because by hypothesis $t \in \mathsf{St}(\Pi) \cup \mathsf{St}(\delta_\Pi(\Pi))$, and $t \in \mathcal{X}$, it must be that $t \in \mathcal{X}(\Pi) \subseteq \mathcal{X}(\mathcal{E}_i)$. Thus, rule (2) is not applicable. Now, according to rule (8), it must be that $\langle \Pi, \phi_1 \rangle \triangleright \delta_\Pi(t) : \tau$. But since $\delta_\Pi(t) \in \mathsf{St}(\delta_\Pi(\Pi))$, we can apply our inductive hypothesis to derive that $\langle \Pi, \phi_2 \rangle \triangleright \delta_\Pi(t) : \tau$. Moreover, the only rule that can then be applied to derive the type of t in the environment $\langle \Pi, \phi_2 \rangle$ is again rule (8) according to which we can conclude that $\langle \Pi, \phi_2 \rangle \triangleright t : \tau$.

Case $t = f(t_1, \ldots, t_n)$ **with** $f \in \mathcal{F}$ **for some terms** $t_1, \ldots, t_n$**.** By inductive hypothesis we know that for all $i \in [\![n]\!]$ and for every type τ_i, $\langle \Pi, \phi_1 \rangle \triangleright t_i : \tau_i$ implies $\langle \Pi, \phi_2 \rangle \triangleright t_i : \tau_i$. Moreover, independently of the formula in the considered typing environment, the only applicable typing rule for t is rule (7) from Figure 2. We can thus conclude that for every type $\tau_1, \ldots, \tau_n$, $\langle \Pi, \phi_1 \rangle \triangleright t : f(\tau_1, \ldots, \tau_n)$ implies $\langle \Pi, \phi_2 \rangle \triangleright t : f(\tau_1, \ldots, \tau_n)$.

(***1*** $\Rightarrow$ ***1'***). Suppose that Π satisfies *1* and consider a formula $\phi \in \mathcal{PS}\text{-LTL}^-$, as well as two indexes $i, j \in [\![\ell]\!]$, such that $e_i = \mathsf{Q}(p, u_1, \ldots, u_n)$ and $e_j = \mathsf{Q}(p', v_1, \ldots, v_m)$. First since *1* is satisfied we have $n = m$ and if we consider $k \in [\![n]\!]$ there must exist a type τ_k such that $\langle \Pi, \mathsf{true} \rangle \triangleright u_k : \tau_k$ and $\langle \Pi, \mathsf{true} \rangle \triangleright v_k : \tau_k$. Now, given that u_k and v_k are in $\mathsf{St}(\Pi)$, according to the statement (†) that we established above, this implies that $\langle \Pi, \phi \rangle \triangleright u_k : \tau_k$ and $\langle \Pi, \phi \rangle \triangleright v_k : \tau_k$ and Π satisfies *1'*.

(***2*** $\Rightarrow$ ***2'***). Suppose that Π satisfies *2*. Given a formula $\phi \in \mathcal{PS}\text{-LTL}^-$ and a term $\mathsf{aenc}(u, v) \in \mathsf{CryptSt}(\Pi)$, *2* tells us that $\langle \Pi, \mathsf{true} \rangle \triangleright v : \alpha$. Now according to the statement (†), this implies that $\langle \Pi, \phi \rangle \triangleright v : \alpha$ which concludes this subproof.

(***3*** $\Rightarrow$ ***3'***). Suppose that Π satisfies *3* and consider a symbolic trace tr of Π, a formula $\phi \in \mathcal{PS}\text{-LTL}^-$ as well as two terms $u, v \in \mathsf{CryptSt}(\mathsf{tr})$, such that $\mathrm{mgu}(u, v) \neq \bot$. According to the definition of a symbolic trace (Definition 6 page 9), there exists two terms $u', v' \in \mathsf{CryptSt}(\Pi)$ such that $u = \mu_{r_1, sid_1}(u')$ and

$v = \mu_{r_2,sid_2}(v')$ for some roles of the protocol r_1, r_2 and some session identifiers sid_1, sid_2. Given that the function μ never associates the same name to two different nonces of the protocol we also get that $\text{mgu}(u', v') \neq \bot$ and because of rules (5) and (9) of Figure 2, this implies that u and u' (*resp.* v and v') are of the same type say τ (*resp.* τ') in the environment $\langle \Pi, \phi \rangle$. By construction of $\mathsf{tr}_{\mathtt{wtn}}(\Pi)$ containing one occurrence of each event of the protocol, and given that function α of $\mathsf{tr}_{\mathtt{wtn}}$ associates the same agent name to each agent variable there must then exist two terms $u'', v'' \in \mathsf{St}(\mathsf{tr}_{\mathtt{wtn}}(\Pi))$, such that $u'' = \mu_{r'_1,sid'_1}(u')$, $v'' = \mu_{r'_2,sid'_2}(v')$, and $\text{mgu}(u'', v'') \neq \bot$ for some roles of the protocol r'_1, r'_2 and some session identifiers sid'_1, sid'_2. Again, because of rules (5) and (9), it must be that $\langle \Pi, \phi \rangle \triangleright u'' : \tau$ and $\langle \Pi, \phi \rangle \triangleright v'' : \tau'$. Now by statement (†) we get that $\langle \Pi, \mathsf{true} \rangle \triangleright u'' : \tau$ and $\langle \Pi, \mathsf{true} \rangle \triangleright v'' : \tau'$, and since $\text{mgu}(u'', v'') \neq \bot$ we get by *3* that $\tau = \tau'$, which concludes the proof of (*3* $\Rightarrow$ *3'*). □

Definition 20 (The class of properties $\Phi(\Pi)$). Let Π be a protocol in $\mathbb{P}$. The class of security properties $\Phi(\Pi)$ is the set of $\mathcal{PS}$-LTL$^-$ formulas ϕ satisfying the three following conditions:

1. $\mathsf{St}(\phi) \subseteq \mathcal{A} \cup \mathcal{X}$ with $\mathcal{X}(\phi) \cap \mathcal{X}(\Pi) = \emptyset$; and
2. ϕ is well-typed in the environment $\langle \Pi, \phi \rangle$; and
3. for all $\mathsf{Q}(t_1, \ldots, t_n) \in \mathsf{Sf}(\phi)$, there exists $\mathsf{Q}(u_1, \ldots, u_n)$ occurring in Π, and if $\langle \Pi, \phi \rangle \triangleright \mathsf{Q} : \tau_1 \times \cdots \times \tau_n$, then for all $i \in [\![n]\!]$ and for all $c \in \mathcal{C}$, $\mathsf{pv}(\alpha), \gamma_c \notin \mathsf{St}(\tau_i)$

It is easy to see that all the interesting secrecy and authentication properties given in Section 4 satisfy these three conditions for all reasonable protocols in $\mathbb{P}$. Indeed, for the authentication properties, all the introduced variables are used within the scope of NC operator or of events whose arguments are all of type α. Moreover, for modeling the secrecy property, each variable appears at most once in a status event (under the Secret predicate), *i.e.* has a unique type. Thus, as long as used with a reasonable protocol (one where status events are introduced as suggested), the properties belong to the above class.

It is important to further note that we only restrict the terms occuring in the property formula to be atomic. However, the protocol specification can include status events that are built using compound terms and agent and term variables. We can thus still specify properties over complex terms such as agreement on compound terms or secrecy of received terms.

We can now state the main result of this work with the following theorem.

Theorem 1. *Let* $\Pi \in \mathbb{P}$ *be a protocol,* T_0 *a set of ground atomic terms and* $\phi \in \Phi(\Pi)$ *a security property. If there exists a symbolic trace* tr *of* Π *and a ground substitution* σ *such that:*

1. $\mathsf{tr}\sigma$ *is a valid execution of* Π w.r.t. *the initial intruder knowledge* T_0*, and*
2. $\langle \mathsf{tr}\sigma, T_0 \rangle \models \neg\phi$*,*

then there also exists a ground substitution σ_{wt} *such that:*

1. $\mathsf{tr}\sigma_{\mathsf{wt}}$ *is a valid execution of* Π w.r.t. *the initial intruder knowledge* T_0*,*
2. $\mathsf{tr}\sigma_{\mathsf{wt}}$ *is well-typed in the environment* $\mathcal{E} = \langle \Pi, \phi \rangle$*, and*
3. $\langle \mathsf{tr}\sigma_{\mathsf{wt}}, T_0 \rangle \models \neg\phi$

Informally, Theorem 1 states that a protocol Π admits an attack on property ϕ if and only if it admits a well-typed attack on ϕ. The interest of this result is that in this case the set of traces to consider for searching an attack is significantly reduced. This will be further illustrated in Corollary 1 page 47.

6. Proofs

This section is dedicated to the proof of the main result of the paper stated in section 5.3. It will first present the background material on unification, constraint systems and simplification of formulas (Section 6.1), then it will go on with some preliminary results and their proofs (Section 6.2), and finally proceed to the actual proof of Theorem 1 (Section 6.3). A reader that is not interested in rather technical and formal details can skip to the next section page 47 for applications of our result.

6.1. Constraint solving and security protocol verification

The proof of Theorem 1 heavily relies on a particular decision procedure for bounded security protocols verification [Cor06, CES06]. This procedure in turn relies on resolution of symbolic constraint systems. Indeed, constraint systems are often used to model and analyse security protocols when a bounded number of sessions is considered (see [Com04, MS01, RT03, CZ06, CDD07]). We thus need to introduce the relative definitions and constraint solving techniques. More precisely, we present the constraint solving procedure of [CDD07]. Only then we will be able to detail the verification algorithm used in our proofs and borrowed from [Cor06, CES06].

6.1.1. Unification

We will only present here unification on the signature $\mathcal{F}$.

Definition 21 (Unification problem). A unification problem P is a set of expressions of the form $t \stackrel{?}{=} u$ where t and u are terms. A unification problem $P = \{t_i \stackrel{?}{=} u_i\}_{i \in [\![h]\!]}$ is said to be in *solved form* if it satisfies the following two conditions.

1. $\forall i \in [\![h]\!],\ t_i \in \mathcal{X}$
2. $\forall i, j \in [\![h]\!],\ j \neq i \ \Rightarrow\ t_i \notin \mathcal{X}(\{t_j, u_j\}) \wedge t_i \notin \mathcal{X}(u_i)$

The unification algorithm presented in [MM82] relies on the set of rules depicted in Figure 3.

$$(a):\ P \cup \{t \stackrel{?}{=} u\} \rightsquigarrow_{\emptyset} P \cup \{u \stackrel{?}{=} t\} \qquad \text{if } u \in \mathcal{X} \text{ and } t \notin \mathcal{X}$$

$$(b):\ P \cup \{t \stackrel{?}{=} t\} \rightsquigarrow_{\emptyset} P \qquad \text{if } t \in (\mathcal{X} \cup \mathcal{A} \cup \mathcal{N} \cup \mathcal{C})$$

$$(c):\ P \cup \{f(t_1, \ldots, t_n) \stackrel{?}{=} f(u_1, \ldots, u_n)\} \rightsquigarrow_{\emptyset} P \cup \{t_1 \stackrel{?}{=} u_1, \ldots, t_n \stackrel{?}{=} u_n\} \quad \text{if } f \in \{\mathsf{pv}, \mathsf{sh}, \langle\rangle, \mathsf{senc}, \mathsf{aenc}, \mathsf{sign}, \mathsf{h}\}$$

$$(d):\ P \cup \{t \stackrel{?}{=} u\} \rightsquigarrow_{\sigma} P\sigma \cup \{t \stackrel{?}{=} u\} \qquad \text{with } \sigma = \{t \mapsto u\} \text{ if } t \in \mathcal{X},\ t \neq u,\ t \notin \mathcal{X}(u) \text{ and } t \text{ occurs in } P$$

Figure 3: Simplification rules

For all $n \geq 1$ we will write $P_0 \rightsquigarrow^n_{\sigma_1 \ldots \sigma_n} P_n$ for the derivation $P_0 \rightsquigarrow_{\sigma_1} P_1 \rightsquigarrow_{\sigma_2} \cdots \rightsquigarrow_{\sigma_n} P_n$. We also introduce the notation $P \rightsquigarrow^*_{\sigma} P'$ to denote that $P \rightsquigarrow^n_{\sigma} P'$ for some $n \geq 1$, or that $\sigma = \emptyset$ and $P = P'$. A unification problem P *admits a solution* if there exists a unification problem P' in solved form such that $P \rightsquigarrow^* P'$.

The following theorem states that the simplification rules applied to a unification problem can be used to effectively compute the most general unifier.

Theorem 2 ([MM82]). *Let t and u be two terms, t and u are unifiable if and only if the unification problem $\{t \stackrel{?}{=} u\}$ admits a solution P. Moreover, if t and u are unifiable, and $P = \{x_1 \stackrel{?}{=} u_1, \ldots, x_n \stackrel{?}{=} u_n\}$ is the solution, unique up to variable renaming, to the unification problem $\{t \stackrel{?}{=} u\}$, then* $\mathrm{mgu}(t, u) = \{x_1 \mapsto u_1, \ldots, x_n \mapsto u_n\}$.

The three following lemmas aim to establish that the most general unifier does not generate new arbitrary subterms or encrypted subterms. For a substitution $\sigma = \mathrm{mgu}(v, w)$ and a term u, (encrypted) subterms of $u\sigma$ either correspond to (encrypted) subterms of u or to (encrypted) subterms of v or w for which this most general unifier has been computed.

Lemma 3. *Let $t, u \in \mathcal{T}$ be two terms, and σ a substitution.*

$$\begin{aligned} t \in \mathsf{CryptSt}(u\sigma) &\ \Rightarrow\ t \in (\mathsf{CryptSt}(u))\sigma \ \vee\ (\exists x \in \mathcal{X}(u),\ t \in \mathsf{CryptSt}(\sigma(x))) \\ t \in \mathsf{St}(u\sigma) &\ \Rightarrow\ t \in (\mathsf{St}(u))\sigma \ \vee\ (\exists x \in \mathcal{X}(u),\ t \in \mathsf{St}(\sigma(x))) \end{aligned}$$

The proof of this lemma is done by induction on the size of term u. When u is a term in $\mathcal{A}$, $\mathcal{Z}$, $\mathcal{N}$, $\mathcal{X}$ or $\mathcal{C}$ the proof is trivial. When $u = \mathsf{f}(v_1, ..., v_n)$ we have either $t = u\sigma$ (impossible for $t \in \mathsf{CryptSt}(u\sigma)$ when $f \notin \{\mathsf{senc}, \mathsf{aenc}, \mathsf{sign}, \mathsf{h}\}$) in which case $t \in (\mathsf{St}(u))\sigma$ (*resp.* $t \in (\mathsf{CryptSt}(u))\sigma$), or $t \in \mathsf{St}(v_i\sigma)$ (*resp.* $t \in \mathsf{CryptSt}(v_i\sigma)$) for some i in which case the induction hypothesis is sufficient to conclude since $\mathcal{X}(v_i) \subseteq \mathcal{X}(u)$, $\mathsf{St}(v_i) \subseteq \mathsf{St}(u)$ and $\mathsf{CryptSt}(v_i) \subseteq \mathsf{CryptSt}(u)$.

Lemma 4. *Let $u, v \in \mathcal{T}$ be two terms, and $\sigma = \mathrm{mgu}(u, v)$. If $\sigma \neq \bot$, then*

$$\forall x \in \mathrm{dom}(\sigma),\ \forall t \in \mathsf{CryptSt}(\sigma(x)),\ \exists w \in \mathsf{CryptSt}(\{u, v\}),\ t = w\sigma$$
$$\forall x \in \mathrm{dom}(\sigma),\ \forall t \in \mathsf{St}(\sigma(x)),\ \exists w \in \mathsf{St}(\{u, v\}),\ t = w\sigma$$

The proof of this lemma works as follows. Let $E_0 = \{u \stackrel{?}{=} v\} \rightsquigarrow^*_{\sigma_1 \cup \cdots \cup \sigma_n} E_n = \{u_1 \stackrel{?}{=} v_1, \ldots, u_n \stackrel{?}{=} v_n\}$ be the derivation of $\sigma = \{u_1 \mapsto v_1, \ldots, u_n \mapsto v_n\}$. We first prove by induction on $i \in [\![n]\!]$ that for all $t \in \mathsf{CryptSt}(E_i)$ (*resp.* $t \in \mathsf{St}(E_i)$), there exists $w \in \mathsf{CryptSt}(\{u, v\})$ (*resp.* $w \in \mathsf{St}(\{u, v\})$) such that $t = w\sigma_1 \ldots \sigma_i$. If $i = 0$ then $\sigma = \{u \mapsto v\}$ and the properties are thus trivially true. For the induction step we need to consider all four simplification rules depicted in Figure 3. For (a), (b) and (c) the result holds since they do not add a new substitution and $\mathsf{CryptSt}(E_i) \subseteq \mathsf{CryptSt}(E_{i-1})$ (*resp.* $\mathsf{St}(E_i) \subseteq \mathsf{St}(E_{i-1})$). Considering rule (d), using Lemma 3, we know that in any case there exists $t' \in \mathsf{CryptSt}(E_{i-1})$ (*resp.* $t' \in \mathsf{St}(E_{i-1})$) such that $t = t'\sigma_i$, and thus by induction hypothesis that there exists $w \in \mathsf{CryptSt}(\{u, v\})$ (*resp.* $w \in \mathsf{St}(\{u, v\})$) such that $t = t'\sigma_i = w\sigma_1 \ldots \sigma_{i-1}\sigma_i$. Finally, let $t \in \mathsf{CryptSt}(\sigma(x))$ (*resp.* $t \in \mathsf{St}(\sigma(x))$) for some $x \in \mathrm{dom}(\sigma)$, then $t \in \mathsf{CryptSt}(E_n)$ (*resp.* $t \in \mathsf{St}(E_n)$), but according to our preliminary result this implies that there exists $w \in \mathsf{CryptSt}(\{u, v\})$ such that $t = w\sigma_1 \ldots \sigma_n = w\sigma$.

Lemma 5. *Let t, u, v, w be four terms such that* $\mathrm{mgu}(v, w) = \sigma \neq \bot$.

$$t \in \mathsf{CryptSt}(u\sigma) \ \Rightarrow\ t \in (\mathsf{CryptSt}(u))\sigma \ \vee\ t \in (\mathsf{CryptSt}(\{v, w\}))\sigma$$
$$t \in \mathsf{St}(u\sigma) \ \Rightarrow\ t \in (\mathsf{St}(u))\sigma \ \vee\ t \in (\mathsf{St}(\{v, w\}))\sigma$$

This lemma is just a corollary of Lemmas 3 and 4 above.

6.1.2. Simplifying constraint systems

Definition 22 (Symbolic constraints). A constraint is an expression of the form $T \Vdash u$, where T is a finite set of terms and u is a term. T is called the *left-hand-side* of the constraint, and u its *right-hand-side*. A *constraint system* is either $\bot$, or a finite set $C = \{T_i \Vdash u_i\}_{i \in [\![n]\!]}$ of constraints satisfying the two following conditions.

- $\forall i \in [\![n-1]\!],\ T_i \subseteq T_{i+1}$
- $\forall i \in [\![n]\!],\ \mathcal{X}(T_i) \subseteq \{x \mid x \in \mathcal{X}(u_j),\ T_j \subsetneq T_i\}$

A ground substitution σ is a solution to C if and only if $T_i\sigma \vdash u_i\sigma$ for all $i \in [\![n]\!]$.

The first condition states that the left-hand-side (the T_i's) are totally ordered, and is often referred to as the *monotonicity condition*, and also implies in the second one that $j < i$. The second condition ensures that any variable

first appears on the right-hand-side (*w.r.t.* the inclusion order of the left-hand-sides), and is often referred to as the *origination property*. C will often be denoted as the conjunction of its constraints.

$$C = \bigwedge_{i \in [\![n]\!]} T_i \Vdash u_i$$

The left-hand-side of C, denoted $\mathrm{lhs}(C)$, is the maximal left-hand-side (*w.r.t* inclusion) constraint in C, *i.e.* $\mathrm{lhs}(C) = \max_{i \in [\![n]\!]}(T_i) = T_n$. The right-hand-side of C, denoted $\mathrm{rhs}(C)$, is the union of the right-hand-sides of its constraints, *i.e.* $\mathrm{rhs}(C) = \bigcup_{i \in [\![n]\!]} \{u_i\}$. The set of variables occurring in C is denoted $\mathcal{X}(C)$ and defined as expected (the last equality between parentheses is due to the origination property).

$$\mathcal{X}(C) = \bigcup_{i \in [\![n]\!]} (\mathcal{X}(T_i) \cup \mathcal{X}(u_i)) \quad (= \bigcup_{i \in [\![n]\!]} \mathcal{X}(u_i))$$

Constraint systems have often been used to define the execution model of protocols. We show how to build, from a scenario sc and an initial intruder knowledge T_0, a symbolic constraint system C, such that the set of solutions to C corresponds to the set of valid executions *w.r.t.* sc and T_0.

Definition 23 ($\mathcal{C}str(\mathsf{tr}, T_0)$). Let Π be a protocol, T_0 a set of atomic terms, sc a scenario of Π, and $\mathsf{tr} = [e_1; \ldots; e_\ell]$ the symbolic trace corresponding to sc. The *symbolic constraint system corresponding to* tr *and* T_0, denoted $\mathcal{C}str(\mathsf{tr}, T_0)$, is defined by:

$$\mathcal{C}str(\mathsf{tr}, T_0) = \{T_0 \cup \mathsf{K}(\mathsf{tr}_{i-1}) \Vdash u \mid e_i = \mathsf{rcv}(p_1, p_2, u),\ i \in [\![\ell]\!],\ p_1, p_2 \in \mathcal{A}\}$$

The left-hand-sides of the resulting constraint system represent the messages sent on the network, while the right-hand-sides correspond to the receptions occurring in tr and thus to the messages that the intruder will have to construct. It is easy to see that the resulting constraint system does satisfy the two conditions of Definition 22.

Example 18. *The constraint system* $\mathcal{C}str(\mathsf{tr}_{\mathsf{Toy}}, T_0)$ *corresponding to the symbolic trace* $\mathsf{tr}_{\mathsf{Toy}}$ *given in Example 4 page 10, with initial knowledge of the intruder* T_0 *contains two constraints corresponding to the two receive events of the trace. These constraints, where* $T_1' = T_0 \cup \{\mathsf{aenc}(\langle \mathsf{aenc}(n^1, b), a\rangle, b)\}$ *are given below.*

$$T_1' \Vdash \mathsf{aenc}(\langle \mathsf{aenc}(x^2, b), \epsilon\rangle, b)$$
$$T_1' \cup \{\mathsf{aenc}(\langle \mathsf{aenc}(x^2, \epsilon), b\rangle, \epsilon)\} \Vdash \mathsf{aenc}(\langle \mathsf{aenc}(x^3, b), \epsilon\rangle, b)$$

The following lemma links constraint systems and the semantics of cryptographic protocols considered in this paper and presented in Section 3.2.

Lemma 6. *Let* Π *be a protocol,* T_0 *a set of ground atomic terms,* sc *a scenario of* Π*, and* tr *the symbolic trace corresponding to* sc*. Let* σ *be a ground substitution such that* $\mathrm{dom}(\sigma) \subseteq \mathcal{X}(\mathsf{tr})$*, then*

$\mathsf{tr}\sigma$ *is a valid execution of* Π w.r.t. T_0 iff σ *is a solution to* $\mathcal{C}str(\mathsf{tr}, T_0)$.

Proof. The proof of this lemma is trivial since the definition of a solution to $\mathcal{C}str(\mathsf{tr}, T_0)$ precisely maps the definition of $\mathsf{tr}\sigma$ being a valid execution of Π *w.r.t.* T_0, namely each receive event in $\mathsf{tr}\sigma$ must be derivable from the knowledge accumulated by the intruder so far.

Definition 24 (Constraint systems in solved form). A constraint system C is said to be in solved form if $C = \bot$, or if $C = \{T_i \Vdash u_i\}_{i \in [\![n]\!]}$ with $u_i \in \mathcal{X}$ for all $i \in [\![n]\!]$.

Such constraint systems are particularly simple since they always admit a solution. Indeed, the substitution $\sigma = \{u_i \mapsto \epsilon\}_{i \in [\![n]\!]}$ is a solution to C if C is in solved form. Note that the empty constraint system is in solved form.

The simplification procedure of a constraint system into a constraint system in solved form used in our proofs relies on the set of simplification rules depicted in Figure 4. All these rules are indexed with a substitution: when it does not appear, it is the substitution with empty domain that is left implicit. For all $n \geq 1$ we write $C_0 \rightsquigarrow_\sigma^n C_n$ for the derivation $C_0 \rightsquigarrow_{\sigma_1} C_1 \rightsquigarrow_{\sigma_2} \cdots \rightsquigarrow_{\sigma_n} C_n$ with $\sigma = \sigma_1\sigma_2 \ldots \sigma_n$. We also introduce the notation $C \rightsquigarrow_\sigma^* D$ to denote that either $C \rightsquigarrow_\sigma^n D$ for some $n \geq 1$, or that $D = C$ and that σ is the empty substitution.

$$\mathsf{R}_1: \quad C \wedge T \Vdash u \quad \rightsquigarrow \quad C \qquad \text{if } T \cup \{x \in \mathcal{X} \mid T' \Vdash x \in C, T' \subsetneq T\} \vdash u$$

$$\mathsf{R}_2: \quad C \wedge T \Vdash u \quad \rightsquigarrow_\sigma \quad C\sigma \wedge T\sigma \Vdash u\sigma$$
$$\text{if } \sigma = \mathrm{mgu}(t, u) \text{ where } t \in \mathsf{St}(T),\ t \neq u,\ t, u \text{ are neither variables nor pairs}$$

$$\mathsf{R}_3: \quad C \wedge T \Vdash u \quad \rightsquigarrow_\sigma \quad C\sigma \wedge T\sigma \Vdash u\sigma$$
$$\text{if } \sigma = \mathrm{mgu}(t_1, t_2),\ t_1, t_2 \in \mathsf{St}(T),\ t_1 \neq t_2,\ t_1, t_2 \text{ are neither variables nor pairs}$$

$$\mathsf{R}_4: \quad C \wedge T \Vdash u \quad \rightsquigarrow_\sigma \quad C\sigma \wedge T\sigma \Vdash u\sigma$$
$$\text{if } \sigma = \mathrm{mgu}(t_2, t_3),\ \mathsf{aenc}(t_1, t_2) \in \mathsf{St}(T),\ \mathsf{pv}(t_3) \in (\mathit{plaintext}(T) \cup \{\mathsf{pv}(\epsilon)\}),\ \ t_2 \neq t_3$$

$$\mathsf{R}_5: \quad C \wedge T \Vdash u \quad \rightsquigarrow \quad \bot \qquad \text{if } \mathcal{X}(T \cup \{u\}) = \emptyset \text{ and } T \not\vdash u$$

$$\mathsf{R}_f: \quad C \wedge T \Vdash f(u_1, \ldots, u_n) \ \rightsquigarrow\ C \wedge \bigwedge_{i \in [n]} T \Vdash u_i \qquad \text{for } f \in \{\langle\rangle, \mathsf{senc}, \mathsf{aenc}, \mathsf{sign}, \mathsf{h}\}$$

Figure 4: Simplification rules for a constraint system

As stated in the following theorem, the set of simplification rules depicted in Figure 4 is correct, complete and terminating: a constraint system C admits a solution θ if and only if there exists a constraint system in solved form D and two substitutions σ and θ' such that $C \rightsquigarrow_\sigma^* D$, θ' is a solution to D, and $\theta = \sigma\theta'$.

It is important to say that for one constraint system that is not in solved form, there exist in general many corresponding constraint systems in solved form. Hence it is useful to define, given a constraint system C the set $\mathrm{sol}(C) = \{(C', \sigma) \mid C \rightsquigarrow_\sigma^* C' \text{ and } C' \text{ in solved form}\}$.

Theorem 3. *Let C be a constraint system not in solved form:*

1. *(Termination) There is no infinite derivation starting from C.*
2. *(Correctness) If there exists a constraint system D and a substitution σ such that $C \rightsquigarrow_\sigma^* D$ and θ' is a solution to D, then $\theta = \sigma\theta'$ is a solution to C.*
3. *(Completeness) If θ is a solution to C, then there exists a constraint system D in solved form as well as two substitutions σ and θ' such that θ' is a solution to D, $\theta = \sigma\theta'$ and $C \rightsquigarrow_\sigma^* D$.*

The paternity of this procedure is attributed to H. Comon-Lundh [Com04]. There exists a certain number of variants to this procedure [CZ06, CDD07, Del07] and for each one, Theorem 3 needs to be reproved. Our work relying on the version of this procedure proposed by V. Cortier *et al.*, we refer the reader to [CDD07] for a proof of this theorem.

Example 19. *Let us consider the constraint system given in Example 18 page 30, with initial knowledge of the intruder T_0. Since in this example no particular knowledge is required from the intruder, we could consider any set T_0 and in particular $T_0 = \emptyset$. We can apply the simplification rules of Figure 4 as follows. First $C = C' \wedge C''$ with $T_1' = T_0 \cup \{\mathsf{aenc}(\langle \mathsf{aenc}(n^1, b), a\rangle, b)\}$,*
$T_2' = T_1' \cup \{\mathsf{aenc}(\langle \mathsf{aenc}(x^2, \epsilon), b\rangle, \epsilon)\}$ *and*
$C' = (T_1' \Vdash \mathsf{aenc}(\langle \mathsf{aenc}(x^2, b), \epsilon\rangle, b))$
$C'' = (T_2' \Vdash \mathsf{aenc}(\langle \mathsf{aenc}(x^3, b), \epsilon\rangle, b))$

By applying twice rule R_f first to an asymmetric encryption and then to a pairing we get:

$C \rightsquigarrow (T_1' \Vdash \langle \mathsf{aenc}(x^2, b), \epsilon\rangle \wedge T_1' \Vdash b \wedge C'')$
$\rightsquigarrow (T_1' \Vdash \mathsf{aenc}(x^2, b) \wedge T_1' \Vdash \epsilon \wedge T_1' \Vdash b \wedge C'') = C_1$

Then, since by definition ϵ and b are deducible from any set of terms, in particular T_1', we can apply rule R_1 to C_1:

$C_1 \rightsquigarrow (T_1' \Vdash \mathsf{aenc}(x^2, b) \wedge T_1' \Vdash b \wedge C'')$
$\rightsquigarrow (T_1' \Vdash \mathsf{aenc}(x^2, b) \wedge C'') = C_2$

Considering $t = \mathsf{aenc}(\langle \mathsf{aenc}(n^1, b), a\rangle, b) \in \mathsf{St}(T_1')$ and $u = \mathsf{aenc}(x^2, b)$ we get $\mathrm{mgu}(t, u) = \{x^2 \mapsto \langle \mathsf{aenc}(n^1, b), a\rangle\} = \sigma$ *and rule R_2 gives:*

$C_2 \rightsquigarrow_\sigma (T_1' \Vdash \mathsf{aenc}(\langle \mathsf{aenc}(n^1, b), a\rangle, b) \wedge C''\sigma) = C_3$

Now, since the right-hand-side of the first constraint of C_3 is included in T_1' and by rule R_1 we obtain:

$C_3 \rightsquigarrow C''\sigma = C_4$

By applying twice rule R_f we get, with $T_3' = T_2'\sigma =$
$T_0\sigma \cup \{\mathsf{aenc}(\langle \mathsf{aenc}(n^1, b), a\rangle, b), \mathsf{aenc}(\langle \mathsf{aenc}(\langle \mathsf{aenc}(n^1, b), a\rangle, \epsilon), b\rangle, \epsilon)\}$*:*

$C_4 \rightsquigarrow T_3' \Vdash \langle \mathsf{aenc}(x^3, b), \epsilon\rangle \wedge T_3' \Vdash b$
$\rightsquigarrow T_3' \Vdash \mathsf{aenc}(x^3, b) \wedge T_3' \Vdash b \wedge T_3' \Vdash \epsilon = C_5$

Like above, given that by definition ϵ and b are deducible from T_3' we obtain:

$C_5 \rightsquigarrow T_3' \Vdash \mathsf{aenc}(x^3, b) \wedge T_3' \Vdash \epsilon$
$\rightsquigarrow T_3' \Vdash \mathsf{aenc}(x^3, b) = C_6$

Finally using $t = \mathsf{aenc}(n^1, b) \in \mathsf{St}(T'_3)$ *and* $u = \mathsf{aenc}(x^3, b)$ *and applying rule* R_2 *with substitution* $\mathrm{mgu}(t, u) = \sigma' = \{x^3 \mapsto n^1\}$ *we get*

$C_6 \rightsquigarrow_{\sigma'} T'_3 \Vdash \mathsf{aenc}(n^1, b)$ *(note that here* $T'_3\sigma' = T'_3$*)*

which in turn can be simplified to the empty constraint system since the right-hand-side can be deduced from the left-hand-side applying successively on $\mathsf{aenc}(\langle \mathsf{aenc}(\langle \mathsf{aenc}(n^1, b), a\rangle, \epsilon), b\rangle, \epsilon)$ *decryption with the known to the intruder private key* $\mathsf{pv}(\epsilon)$*, depairing, decryption again and depairing again.*

6.1.3. Simplifying $\mathcal{PS}$-LTL formulas

So far our security properties have been modeled using temporal logic and dedicated status events. The goal of this part of the paper is to show how a $\mathcal{PS}$-LTL$^+$ formula can be transformed into an *elementary* formula whose validity will be easier to verify. This new formula will no more contain temporal operators but instead (dis)equalities on terms and symbolic constraints, on which the previously defined simplification method can be applied.

More precisely, the approach proposed by R. Corin [Cor06, CES06] can be split into two steps. Let ϕ be the considered formula in $\mathcal{PS}$-LTL$^-$. The first step consists, given a protocol and a symbolic trace, in translating the corresponding attack formula $\neg\phi \in \mathcal{PS}$-LTL$^+$ into an equivalent *elementary* formula π. This translation relies on the simplification function $\mathbf{T}$ defined below. The second step consists in using the elementary formula, together with (the constraint system associated to) a symbolic trace, as parameters for a decision procedure that will tell whether the given trace can correspond to an attack on the original formula.

Definition 25 (Elementary formulas). The set of elementary formulas is defined by the following grammar:

$$\pi \quad ::= \quad \mathsf{true} \quad | \quad t_1 = t_2 \quad | \quad T \Vdash t \quad | \quad \neg\pi \quad | \quad \pi \vee \pi \quad | \quad \exists x, \pi$$

Let π be an elementary formula. We distinguish the set of free variables of π denoted $\mathsf{Free}_r(\pi)$ appearing on the right of the $=$ or $\Vdash$ symbols, from the set of free variables of π denoted $\mathsf{Free}_l(\pi)$ appearing on the left of the $=$ or $\Vdash$ symbols.

$\mathsf{Free}_l(\mathsf{true}) = \emptyset$	$\mathsf{Free}_r(\mathsf{true}) = \emptyset$
$\mathsf{Free}_l(t_1 = t_2) = \mathcal{X}(t_1)$	$\mathsf{Free}_r(t_1 = t_2) = \mathcal{X}(t_2)$
$\mathsf{Free}_l(T \Vdash t) = \mathcal{X}(T)$	$\mathsf{Free}_r(T \Vdash t) = \mathcal{X}(t)$
$\mathsf{Free}_l(\neg\pi) = \mathsf{Free}_l(\pi)$	$\mathsf{Free}_r(\neg\pi) = \mathsf{Free}_r(\pi)$
$\mathsf{Free}_l(\pi_1 \vee \pi_2) = \mathsf{Free}_l(\pi_1) \cup \mathsf{Free}_l(\pi_2)$	$\mathsf{Free}_r(\pi_1 \vee \pi_2) = \mathsf{Free}_r(\pi_1) \cup \mathsf{Free}_r(\pi_2)$
$\mathsf{Free}_l(\exists x,\ \pi) = \mathsf{Free}_l(\pi) \setminus \{x\}$	$\mathsf{Free}_r(\exists x,\ \pi) = \mathsf{Free}_r(\pi) \setminus \{x\}$

Definition 26 ($\sigma \models' \pi$). Let π be an elementary formula and σ a closed substitution such that $\mathsf{Free}_r(\pi) = \emptyset$ and $\mathsf{Free}_l(\pi) = \mathrm{dom}(\sigma)$. The validity of an elementary formula π, denoted $\sigma \models' \pi$ is inductively defined as follows:

$$
\begin{array}{lll}
\sigma \models' \mathsf{true} & & \\
\sigma \models' t_1 = t_2 & \textit{iff} & t_1\sigma = t_2 \\
\sigma \models' T \Vdash t & \textit{iff} & T\sigma \vdash t \\
\sigma \models' \neg\pi & \textit{iff} & \neg\sigma \models' \pi \\
\sigma \models' \pi_1 \vee \pi_2 & \textit{iff} & \sigma \models' \pi_1 \textit{ or } \sigma \models' \pi_2 \\
\sigma \models' \exists x,\ \pi & \textit{iff} & \exists t \in \mathcal{T} \textit{ such that } \sigma \models' \pi\{x \mapsto t\}
\end{array}
$$

Definition 27 (Translation T). Let ϕ be a formula of $\mathcal{PS}$-LTL$^+$. Let also Π be a protocol, tr a trace of Π, *i.e.* corresponding to some scenario of Π, and T_0 a set of ground atoms. $\mathbf{T}(\phi, \mathsf{tr}, T_0)$ denotes the translation of ϕ in an "equivalent" (in the sense that will be established by Lemma 7 page 35) elementary formula and is defined as follows:

$$
\begin{array}{rcl}
\mathbf{T}(\mathsf{true}, \mathsf{tr}, T_0) & \rightarrow & \mathsf{true} \\
\mathbf{T}(\mathsf{learn}(t), \mathsf{tr}, T_0) & \rightarrow & T_0 \cup \mathsf{K}(\mathsf{tr}) \Vdash t \\
\mathbf{T}(\mathsf{Q}(t_1, \ldots, t_n), \mathsf{tr}, T_0) & \rightarrow & \left\{ \begin{array}{ll} t'_1 = t_1 \wedge \cdots \wedge t'_n = t_n & \text{if } \mathsf{tr} = \mathsf{tr}'; \mathsf{Q}(t'_1, \ldots, t'_n) \\ \neg\mathsf{true} & \text{otherwise} \end{array} \right. \\
\mathbf{T}(\mathsf{C}(t), \mathsf{tr}, T_0) & \rightarrow & \epsilon = t \vee \bigvee\limits_{\mathsf{pv}(u) \in T_0} u = t \vee \bigvee\limits_{\mathsf{sh}(u,t) \in T_0} \epsilon \neq u \\
\mathbf{T}(\neg\phi, \mathsf{tr}, T_0) & \rightarrow & \neg\mathbf{T}(\phi, \mathsf{tr}, T_0) \\
\mathbf{T}(\phi_1 \vee \phi_2, \mathsf{tr}, T_0) & \rightarrow & \mathbf{T}(\phi_1, \mathsf{tr}, T_0) \vee \mathbf{T}(\phi_2, \mathsf{tr}, T_0) \\
\mathbf{T}(\mathcal{Y}\phi, \mathsf{tr}, T_0) & \rightarrow & \left\{ \begin{array}{ll} \mathbf{T}(\phi, \mathsf{tr}', T_0) & \text{if } \mathsf{tr} = \mathsf{tr}'; e \\ \neg\mathsf{true} & \text{otherwise} \end{array} \right. \\
\mathbf{T}(\phi_1\ \mathcal{S}\ \phi_2, \mathsf{tr}, T_0) & \rightarrow & \left\{ \begin{array}{ll} \mathbf{T}(\phi_2, \mathsf{tr}, T_0)\ \vee\ (\mathbf{T}(\phi_1\ \mathcal{S}\ \phi_2, \mathsf{tr}', T_0) \wedge \mathbf{T}(\phi_1, \mathsf{tr}, T_0)) & \\ & \text{if } \mathsf{tr} = \mathsf{tr}'; e \\ \neg\mathsf{true} & \text{otherwise} \end{array} \right. \\
\mathbf{T}(\exists x,\ \phi, \mathsf{tr}, T_0) & \rightarrow & \exists x,\ \mathbf{T}(\phi, \mathsf{tr}, T_0)
\end{array}
$$

Example 20. *To illustrate this translation, let us consider the trace* $\mathsf{tr}_{\mathsf{Toy}}$ *introduced in Example 4 page 10 to which we have added the status events as suggested in Example 7 page 13 to verify the secrecy of generated nonces.*

$$
\mathsf{tr}'_{\mathsf{Toy}} = \left[\begin{array}{l}
\mathsf{snd}(a, b, \mathsf{aenc}(\langle \mathsf{aenc}(n^1, b), a \rangle, b)); \\
\mathsf{Secret}(a, a, b, n^1); \\
\mathsf{rcv}(b, \epsilon, \mathsf{aenc}(\langle \mathsf{aenc}(x^2, b), \epsilon \rangle, b)); \\
\mathsf{snd}(b, \epsilon, \mathsf{aenc}(\langle \mathsf{aenc}(x^2, \epsilon), b \rangle, \epsilon)); \\
\mathsf{rcv}(b, \epsilon, \mathsf{aenc}(\langle \mathsf{aenc}(x^3, b), \epsilon \rangle, b)); \\
\mathsf{snd}(b, \epsilon, \mathsf{aenc}(\langle \mathsf{aenc}(x^3, \epsilon), b \rangle, \epsilon))
\end{array} \right]
$$

As said in Example 19 page 32, the constraint system $\mathcal{C}str(\mathsf{tr}, T_0) = \mathcal{C}str(\mathsf{tr}'_{\mathsf{Toy}}, T_0)$ *can be simplified into an empty constraint system, more precisely* $\mathcal{C}str(\mathsf{tr}, T_0) \rightsquigarrow^*_\sigma C = \emptyset$ *with* $\sigma = \{x^2 \mapsto \langle \mathsf{aenc}(n^1, b), a \rangle, x^3 \mapsto n^1\}$*. Applying this substitution to* $\mathsf{tr}'_{\mathsf{Toy}}$ *gives:*

$$
\mathsf{tr}''_{\mathsf{Toy}} = \left[\begin{array}{l}
\mathsf{snd}(a, b, \mathsf{aenc}(\langle \mathsf{aenc}(n^1, b), a \rangle, b)); \\
\mathsf{Secret}(a, a, b, n^1); \\
\mathsf{rcv}(b, \epsilon, \mathsf{aenc}(\langle \mathsf{aenc}(\langle \mathsf{aenc}(n^1, b), a \rangle, b), \epsilon \rangle, b)); \\
\mathsf{snd}(b, \epsilon, \mathsf{aenc}(\langle \mathsf{aenc}(\langle \mathsf{aenc}(n^1, b), a \rangle, \epsilon), b \rangle, \epsilon)); \\
\mathsf{rcv}(b, \epsilon, \mathsf{aenc}(\langle \mathsf{aenc}(n^1, b), \epsilon \rangle, b)); \\
\mathsf{snd}(b, \epsilon, \mathsf{aenc}(\langle \mathsf{aenc}(n^1, \epsilon), b \rangle, \epsilon))
\end{array} \right]
$$

Let us also consider the $\mathcal{PS}$-LTL$^+$ attack formula

$$\phi = \exists y, \exists y_1, \exists y_2, \exists z, [\Diamond(\mathsf{Secret}(y, y_1, y_2, z)) \wedge \mathsf{NC}(y_1) \wedge \mathsf{NC}(y_2) \wedge \mathsf{learn}(z)]$$

corresponding to the negation of the one introduced in Example 7. We will now apply the translation $\mathbf{T}(\phi, \mathsf{tr}''_{\mathsf{Toy}}, T_0)$ with $T_0 = \emptyset$. First if $\phi' = \Diamond(\mathsf{Secret}(y, y_1, y_2, z)) \wedge \mathsf{NC}(y_1) \wedge \mathsf{NC}(y_2) \wedge \mathsf{learn}(z)$ *we have*

$$\begin{array}{rcl} \mathbf{T}(\phi, \mathsf{tr}''_{\mathsf{Toy}}, T_0) = & \exists y, \exists y_1, \exists y_2, \exists z, \mathbf{T}(\phi', \mathsf{tr}''_{\mathsf{Toy}}, T_0) \\ = & \exists y, \exists y_1, \exists y_2, \exists z, [\mathbf{T}(\Diamond(\mathsf{Secret}(y, y_1, y_2, z)), \mathsf{tr}''_{\mathsf{Toy}}, T_0) \\ & \wedge \mathbf{T}(\mathsf{NC}(y_1), \mathsf{tr}''_{\mathsf{Toy}}, T_0) \wedge \mathbf{T}(\mathsf{NC}(y_2), \mathsf{tr}''_{\mathsf{Toy}}, T_0) \\ & \wedge \mathbf{T}(\mathsf{learn}(z), \mathsf{tr}''_{\mathsf{Toy}}, T_0)] \end{array}$$

Now since T_0 does not contain private keys, nor keys shared by other participants, we have
$\mathbf{T}(\mathsf{NC}(y_1), \mathsf{tr}''_{\mathsf{Toy}}, T_0) = (\epsilon \neq y_1)$

The translation of the learn *operator gives a constraint system.*
$\mathbf{T}(\mathsf{learn}(z), \mathsf{tr}''_{\mathsf{Toy}}, T_0) = T_0 \cup \mathsf{K}(\mathsf{tr}''_{\mathsf{Toy}}) \Vdash z$

And the translation of the once operator gives, with $\mathsf{tr}''_{\mathsf{Toy}} = \mathsf{tr}'; e$
$\mathbf{T}(\Diamond(\mathsf{Secret}(y, y_1, y_2, z)), \mathsf{tr}''_{\mathsf{Toy}}, T_0) = \mathbf{T}(\mathsf{Secret}(y, y_1, y_2, z), \mathsf{tr}''_{\mathsf{Toy}}, T_0) \ \vee$
$(\mathbf{T}(\mathsf{true}\ \mathcal{S}\ \mathsf{Secret}(y, y_1, y_2, z), \mathsf{tr}', T_0) \wedge \mathsf{true}$

By iteratively applying this translation to prefixes of $\mathsf{tr}''_{\mathsf{Toy}}$ each time shorter we eventually get with $\mathsf{tr}'' = \mathsf{snd}(a, b, \mathsf{aenc}(\langle \mathsf{aenc}(n^1, b), a \rangle, b)); \mathsf{Secret}(a, a, b, n^1)$

$$\begin{array}{rcl} \mathbf{T}(\Diamond(\mathsf{Secret}(y, y_1, y_2, z)), \mathsf{tr}''_{\mathsf{Toy}}, T_0) = & \mathbf{T}(\mathsf{Secret}(y, y_1, y_2, z), \mathsf{tr}'', T_0) \\ = & (a = y \wedge a = y_1 \wedge b = y_2 \wedge n^1 = z) \end{array}$$

The elementary formula corresponding to ϕ on trace $\mathsf{tr}''_{\mathsf{Toy}}$ is then
$\pi = \exists y, \exists y_1, \exists y_2, \exists z, [a = y \wedge a = y_1 \wedge b = y_2 \wedge n^1 = z \wedge \epsilon \neq y_1 \wedge \epsilon \neq y_2 \wedge T_0 \cup \mathsf{K}(\mathsf{tr}''_{\mathsf{Toy}}) \Vdash z]$

The following lemma states that the translation $\mathbf{T}$ is correct.

Lemma 7. *[Cor06, CES06] Let ϕ be a closed formula of $\mathcal{PS}$-LTL$^+$, tr a trace, T_0 a set of closed atomic terms, and σ a ground substitution such that $\mathcal{X}(\mathsf{tr}) \subseteq \mathrm{dom}(\sigma)$.*

$$\langle \mathsf{tr}\sigma, T_0 \rangle \models \phi \text{ iff } \sigma \models' \mathbf{T}(\phi, \mathsf{tr}, T_0)$$

Moreover, no atomic formula of the form $T \Vdash t$ occurs negatively in $\mathbf{T}(\phi, \mathsf{tr}, T_0)$, i.e. *under an odd number of negations.*

The proof of the above equivalence, given in [Cor06], is made by induction on the number of rewriting steps used to compute $\mathbf{T}(\phi, \mathsf{tr}, T_0)$. Proving that constraints do not appear negatively in $\mathbf{T}(\phi, \mathsf{tr}, T_0)$, can also be done by induction. Indeed, these constraints appear when translating learn operators of the original formula. Moreover these learn could not appear negatively in a $\mathcal{PS}$-LTL$^+$ formula and the translation preserves the number and order of negations.

6.1.4. Decision procedure

We now present a decision procedure that, given a symbolic trace tr, a set of ground atomic terms T_0 and a formula ϕ in $\mathcal{PS}$-LTL$^+$, decides if there exists a ground substitution σ such that $\langle \mathsf{tr}\sigma, T_0 \rangle \models \phi$. The approach presented here is the one proposed in [Cor06, CES06]. It consists of two steps: *(i)* ϕ is first simplified in an equivalent elementary formula (in the sense of Lemma 7) $\pi = \mathbf{T}(\phi, \mathsf{tr}, T_0)$, and then *(ii)* π is passed as an argument to the decision procedure $\mathbf{D}$ that we will now describe.

Let us note that ϕ being a formula in $\mathcal{PS}$-LTL$^+$, π is necessarily a positive elementary formula of the form $\pi = \exists x_1, \ldots \exists x_n,\ \pi'$. The quantifier free formula π' can easily be put in disjunctive normal form, and can be written $\pi' = \bigvee_{i \in [\![k]\!]} \pi'_i$, and for all $i \in [\![k]\!]$, π'_i is of the form $\pi'_i = C_i \wedge Eq_i \wedge Deq_i$ with:

- $C_i = \{T_j \Vdash t_j\}_{j \in [\![\ell_i]\!]}$
- $Eq_i = \{s_j = s'_j\}_{j \in [\![m_i]\!]}$
- $Deq_i = \{u_j \neq u'_j\}_{j \in [\![n_i]\!]}$

Due to the translation $\mathbf{T}$, each π'_i is a conjunction of (negations of) constraints and equalities, and due to the fact that ϕ is a $\mathcal{PS}$-LTL$^+$ formula, constraints in π can only occur positively in π according to Lemma 7, hence the possible reordering of the formula π as stated above.

Let C be a constraint system in solved form and $\pi = \bigvee_{i \in [\![k]\!]} (C_i \wedge Eq_i \wedge Deq_i)$ a positive elementary formula, we can apply the Procedure 1 below. This procedure introduces a substitution called σ_F, used in the first **return** instruction, that just maps every uninstantiated variable of $((\mathsf{tr}\sigma)\theta)\rho$ to a distinct *fresh* (*i.e.* not used yet) constant in the initial intruder knowledge T_0.

Procedure 1 $\mathbf{D}(C, \pi)$

for $i = 1$ to k **do**
  $\theta \leftarrow \mathrm{mgu}(Eq_i)$
  if $\theta \neq \bot$ **then**
    $E \leftarrow C_i\theta \wedge C\theta$
    $\Sigma \leftarrow \{\rho \mid E \rightsquigarrow^*_\rho F$ and F in solved form$\}$
    for $\rho \in \Sigma$ **do**
      if $(Deq_i)\theta\rho \equiv \mathsf{true}$ **then**
        return $\theta\rho\sigma_F$
      end if
    end for
  end if
end for
return $\bot$

The following lemma establishes the correctness of the procedure $\mathbf{D}$. We refer the reader to [Cor06, CES06] for a detailed proof.

Lemma 8. *Let* Π *be a protocol,* sc *a scenario of* Π*,* T_0 *a set of ground atomic terms,* tr *the symbolic trace corresponding to* sc*,* ϕ *a formula in* $\mathcal{PS}$*-LTL*$^+$*. Let* C *be a constraint system in solved form and* σ *a substitution such that* $\mathcal{C}str(\mathsf{tr}, T_0) \rightsquigarrow_\sigma^* C$*. Let* π *be the elementary formula such that* $\pi = \mathbf{T}(\phi, \mathsf{tr}\sigma, T_0)$*, and* π' *the corresponding positive elementary formula in disjunctive normal form.*

1. *if* $\mathbf{D}(C, \pi) = \theta$ $(\neq \bot)$*, then* $\theta \models' \pi$ *and* $\mathsf{tr}\sigma\theta$ *is a valid execution of* Π*,* w.r.t *the initial intruder knowledge* T_0*;*
2. *if* $\gamma \models' \pi$ *and* $\mathsf{tr}\sigma\gamma$ *is a valid execution of* Π*,* w.r.t *the initial intruder knowledge* T_0 *for some substitution* γ*, then there exists a substitution* θ *such that* $\mathbf{D}(C, \pi) = \theta$ $(\neq \bot)$*.*

More precisely, in the second item, as $\mathbf{D}$ does not instantiate all the variables of tr, γ is an instance of θ, in other words there exists a substitution γ' such that $\gamma = \theta\gamma'$.

Example 21. *Following the translation of Example 20 page 34, we have*
$\pi = \exists y, \exists y_1, \exists y_2, \exists z,$

$a = y \wedge a = y_1 \ \wedge \ b = y_2 \ \wedge \ n^1 = z \ \wedge \ \epsilon \neq y_1 \ \wedge \ \epsilon \neq y_2 \ \wedge \ T_0 \cup \mathsf{K}(\mathsf{tr}\sigma) \Vdash z$

Here the quantifier free part of π *is already in disjunctive normal form with* $k = 1$*.*

Since the constraint system in solved form associated to trace $\mathsf{tr}'_{\mathsf{Toy}}$ *of Example 20 is empty, we compute* $\mathbf{D}(\emptyset, \pi)$*. The procedure then considers* $\theta = \mathrm{mgu}\{a = y, a = y_1, b = y_2, n^1 = z\} = \{y \mapsto a, y^1 \mapsto a, y^2 \mapsto b, z \mapsto n^1\}$*. And assigns to* E *the constraint system* $C_1\theta$ *(since* C *is empty) with* $E = C_1\theta = (\mathsf{K}(\mathsf{tr}\sigma) \Vdash z)\theta$
$= \{\mathsf{aenc}(\langle \mathsf{aenc}(n^1, b), a\rangle, b)), \mathsf{aenc}(\langle \mathsf{aenc}(\langle \mathsf{aenc}(n^1, b), a\rangle, \epsilon), b\rangle, \epsilon),$
$\mathsf{aenc}(\langle \mathsf{aenc}(n^1, \epsilon), b\rangle, \epsilon)\} \Vdash n^1$

Now applying rule R_1 *to* E *gives an empty constraint system,* F *thus in solved form with empty substitution* ρ*. Since there are no uninstantiated variables in* $\mathsf{tr}\sigma\theta$ *we do not need to define* σ_F *and procedure* $\mathbf{D}$ *returns substitution* θ*.*

The following theorem states that the two-steps verification procedure consisting in first applying the transformation $\mathbf{T}$, and then calling procedure $\mathbf{D}$ on the resulting formula is correct and complete. The reader can consult [Cor06, CES06] for a detailed proof of this result.

Theorem 4. *Let* Π *be a protocol,* sc *a scenario of* Π*,* T_0 *a set of ground atomic terms,* tr *the symbolic trace corresponding to* sc*,* ϕ *a formula in* $\mathcal{PS}$*-LTL*$^-$*. Let* $A_\phi = \neg\phi$ *be the corresponding attack formula. Let* $C = \mathcal{C}str(\mathsf{tr}, T_0)$*.*

$$\langle \mathsf{tr}, T_0\rangle \models \phi \qquad \text{iff} \qquad \forall (C', \sigma) \in \mathrm{sol}(C),\ \mathbf{D}(C', \pi) = \bot$$

for π *the disjunctive normal form of* $\mathbf{T}(A_\phi, \mathsf{tr}\sigma, T_0)$*.*

6.2. Some preliminary results

Before proceeding to the proof of Theorem 1 page 27, we first state and prove a few preliminary results that will be useful for the main proof, which can be found on page 43.

Definition 28 (Well-typed set of terms). Let $\mathcal{E} = \langle \Pi, \phi \rangle$ be a typing environment such that Π and ϕ are well-typed in the environment $\mathcal{E}$, and T a set of terms. T is said to be well-typed in the typing environment $\mathcal{E}$, if it satisfies the following two conditions:

1. $\forall u, v \in \mathsf{CryptSt}(T),\ \mathrm{mgu}(u, v) \neq \bot \ \Rightarrow\ \exists \tau,\ \mathcal{E} \rhd u : \tau \ \wedge\ \mathcal{E} \rhd v : \tau$, and
2. $\forall \mathsf{aenc}(u, v) \in \mathsf{CryptSt}(T),\ \mathcal{E} \rhd v : \alpha$

Proposition 2. *Given a typing environment $\mathcal{E} = \langle \Pi, \phi \rangle$ such that Π and ϕ are well-typed in the environment $\mathcal{E}$, and two terms u, v of the same type,* i.e. $\exists \tau,\ \mathcal{E} \rhd u : \tau \ \wedge\ \mathcal{E} \rhd v : \tau$. *Then either* $\mathrm{mgu}(u, v)$ *is well-typed or* $\mathrm{mgu}(u, v) = \bot$.

Proof. Suppose that $\mathrm{mgu}(u, v) \neq \bot$. According to Theorem 2 page 28, the substitution $\mathrm{mgu}(u, v)$ can be computed applying the unification algorithm described at Section 6.1.1 to the initial set of equations $E_0 = \{u \stackrel{?}{=} v\}$. In other words, there exists a simplification sequence $E_0 \rightsquigarrow^k E_k = \{x_1 \stackrel{?}{=} t_1, \ldots, x_m \stackrel{?}{=} t_m\}$ such that $\mathrm{mgu}(u, v) = \{x_1 \mapsto t_1, \ldots, x_m \mapsto t_m\}$.

We will say that a set of equations $\{u_i \stackrel{?}{=} v_i\}_{i \in [\![n]\!]}$ is well-typed in the environment $\mathcal{E}$ if for all $i \in [\![n]\!]$, u_i and v_i are of the same type, *i.e.* there exists a type τ_i such that $\mathcal{E} \rhd u_i : \tau_i$ and $\mathcal{E} \rhd v_i : \tau_i$.

We now show by induction on $j \in [\![k]\!]$ that E_j is well-typed in the environment $\mathcal{E}$.

Base case ($j = 0$). By hypothesis we have that u and v are of the same type τ in the environment $\mathcal{E}$, and thus E_0 is well-typed in the environment $\mathcal{E}$.

Inductive case ($j > 0$). By inductive hypothesis we have that for all $h < j$, E_h is well-typed in the environment $\mathcal{E}$. In order to establish that E_j is also well-typed in the environment $\mathcal{E}$, we proceed by case analysis on the rule R involved in the simplification $E_{j-1} \rightsquigarrow E_j$.

Case $\mathsf{R} = (a)$. In that case, $E_{j-1} = E \cup \{t \stackrel{?}{=} w\}$, and $E_j = E \cup \{w \stackrel{?}{=} t\}$ for some set of equations E and some terms t and w. It is then obvious that if E_{j-1} is well-typed in the environment $\mathcal{E}$, so is E_j.

Case $\mathsf{R} = (b)$. In that case, $E_{j-1} = E \cup \{t \stackrel{?}{=} t\}$, and $E_j = E$ for some set of equations E and some term t. It is again obvious that if E_{j-1} is well-typed in the environment $\mathcal{E}$, so is $E = E_j$.

Case $\mathsf{R} = (c)$. In that case, $E_{j-1} = E \cup \{f(t_1, \ldots, t_h) \stackrel{?}{=} f(w_1, \ldots, w_h)\}$, and $E_j = E \cup \{t_1 \stackrel{?}{=} w_1, \ldots, t_h \stackrel{?}{=} w_h\}$ for some set of equations E and some terms t_1, w_1, …, t_h, w_h. Now, because by hypothesis we have that E_{j-1} is well-typed in the environment $\mathcal{E}$, we know that E is well-typed in the environment $\mathcal{E}$, and that there exists a type τ such that $\mathcal{E} \rhd f(t_1, \ldots, t_h) : \tau$ and $\mathcal{E} \rhd f(w_1, \ldots, w_h) : \tau$.

Then, by inspection of our typing system (Figure 2 page 21) we know that for some types $\tau_1, \ldots, \tau_h$, $\tau = f(\tau_1, \ldots, \tau_h)$, but also that for all $i \in [\![h]\!]$, $\mathcal{E} \triangleright t_i : \tau_i$ and $\mathcal{E} \triangleright w_i : \tau_i$. Hence, $E_j = E \cup \{t_1 \stackrel{?}{=} w_1, \ldots, t_h \stackrel{?}{=} w_h\}$ is the union of two sets of equations, well-typed in the environment $\mathcal{E}$, and is thus also well-typed in the environment $\mathcal{E}$.
Case $\mathsf{R} = (d)$**.** In that case, $E_{j-1} = E \cup \{t \stackrel{?}{=} w\}$, and $E_j = E\{t \mapsto w\} \cup \{t \stackrel{?}{=} w\}$ for some set of equations E and some terms t and w. Since E and $\{t \stackrel{?}{=} w\}$ are included in E_{j-1} and E_{j-1} is well-typed in the environment $\mathcal{E}$, so are E and the substitution $\{t \mapsto w\}$. But then Lemma 2 page 23 implies that $E\{t \mapsto w\}$ is also well-typed in the environment $\mathcal{E}$, and thus $E_j = E\{t \mapsto w\} \cup \{t \stackrel{?}{=} w\}$ is well-typed in the environment $\mathcal{E}$.

At this point, we have demonstrated that for all $i \in [\![m]\!]$, x_i and t_i are of the same type in the environment $\mathcal{E}$, and thus that the substitution $\mathrm{mgu}(u, v) = \{x_1 \mapsto t_1, \ldots, x_m \mapsto t_m\}$ is well-typed in the environment $\mathcal{E}$. □

Proposition 3. *Let $\mathcal{E} = \langle \Pi, \phi \rangle$ be a typing environment such that Π and ϕ are well-typed in the environment $\mathcal{E}$, T a set of terms that is well-typed in $\mathcal{E}$, and $v, w \in \mathsf{St}(T)$ two terms of the same type. Then either $\sigma = \mathrm{mgu}(v, w) = \bot$ or $T\sigma$ is well-typed in $\mathcal{E}$.*

Proof. Let $\mathcal{E} = \langle \Pi, \phi \rangle$ be a typing environment such that Π and ϕ are well-typed in the environment $\mathcal{E}$, T a set of terms that is well-typed in $\mathcal{E}$, and $v, w \in \mathsf{St}(T)$ two terms such that $\sigma = \mathrm{mgu}(v, w) \neq \bot$.

1. Let $t, u \in \mathsf{CryptSt}(T\sigma)$ such that $\mathrm{mgu}(t, u) \neq \bot$. In that case, there exists two terms $t', u' \in T$ such that $t \in \mathsf{CryptSt}(t'\sigma)$ and $u \in \mathsf{CryptSt}(u'\sigma)$, and according to Lemma 5 page 29 it is then the case that

$$t \in \mathsf{CryptSt}(t')\sigma \ \vee \ t \in \mathsf{CryptSt}(\{v, w\})\sigma$$

and that

$$u \in \mathsf{CryptSt}(u')\sigma \ \vee \ u \in \mathsf{CryptSt}(\{v, w\})\sigma$$

We now consider the two possible cases for t. If $t \in \mathsf{CryptSt}(t')\sigma$ then there exists $t'' \in \mathsf{CryptSt}(t') \subseteq \mathsf{CryptSt}(T)$ such that $t = t''\sigma$. The other case where $t \in \mathsf{CryptSt}(\{v, w\})\sigma$ similarly gives a $t'' \in \mathsf{CryptSt}(\{v, w\}) \subseteq \mathsf{CryptSt}(T)$ such that $t = t''\sigma$. The same reasoning for u gives that there exists a $u'' \in \mathsf{CryptSt}(T)$ such that $u = u''\sigma$. Now by hypothesis we have that $\mathrm{mgu}(t, u) \neq \bot$ (let $\theta = \mathrm{mgu}(t, u)$). Then $t\theta = (t''\sigma)\theta = (u''\sigma)\theta = u\theta$, and thus t'' and u'' are unifiable which in turn implies that $\mathrm{mgu}(t'', u'') \neq \bot$. Now, let's remind ourselves that $t'', u'' \in \mathsf{CryptSt}(T)$ and thus by hypothesis on T (T is well-typed in the environment $\mathcal{E}$) we know that t'' and u'' are of the same type in the environment $\mathcal{E}$, *i.e.* $\exists \tau, \ \mathcal{E} \triangleright t'' : \tau \ \wedge \ \mathcal{E} \triangleright u'' : \tau$. Moreover, v and w being of the same type, we know according to Proposition 2 page 38, that σ is well-typed, and thus according to Lemma 2 page 23 that $t (= t''\sigma)$ and t'' (*resp.* $u = (u''\sigma)$ and u'') are of the same type, which allows us to conclude that $\mathcal{E} \triangleright t : \tau$ and $\mathcal{E} \triangleright u : \tau$.

2. Let $\mathsf{aenc}(t,u) \in \mathsf{CryptSt}(T\sigma)$. In that case, there exists a term $t' \in T$ such that $\mathsf{aenc}(t,u) \in \mathsf{CryptSt}(t'\sigma)$, and according to Lemma 5 page 29 it is the case that

$$\mathsf{aenc}(t,u) \in \mathsf{CryptSt}(t')\sigma \ \vee \ \mathsf{aenc}(t,u) \in \mathsf{CryptSt}(\{v,w\})\sigma$$

Using the same arguments as the ones presented in the previous item, we get that $\mathsf{aenc}(t,u) = v'\sigma$ with v' belonging either to $\mathsf{CryptSt}(t')$, $\mathsf{CryptSt}(v)$ or $\mathsf{CryptSt}(w)$, all included in $\mathsf{CryptSt}(T)$ and consequently $v' = \mathsf{aenc}(t'',u'')$ for some terms t'' and u''. Now, since v belongs to $\mathsf{CryptSt}(T)$ where T is by hypothesis well-typed in the environment $\mathcal{E}$, we know that $\mathcal{E} \triangleright u'' : \alpha$. Moreover, v and w being of the same type, we know according to Proposition 2, that σ is well-typed, and thus according to Lemma 2 that $u = (u''\sigma)$ and u'' are of the same type, which allows us to conclude that $\mathcal{E} \triangleright u : \alpha$.

We have shown that $T\sigma$ satisfies the two conditions of Definition 28 page 38, and can thus conclude that $T\sigma$ is well-typed. □

Proposition 4. *Let $\mathcal{E} = \langle \Pi, \phi \rangle$ be a typing environment such that Π and ϕ are well-typed in the environment $\mathcal{E}$, and T a set of terms well-typed in the environment $\mathcal{E}$ such that $\mathcal{Z}(T) = \emptyset$. Let C and D be two constraint systems and σ a substitution such that $\mathrm{lhs}(C) \subseteq T$ and $\mathrm{rhs}(C) \subseteq \mathsf{St}(T)$, and such that D admits a solution. If $C \rightsquigarrow_\sigma^n D$, then*

- *σ is well-typed in the environment $\mathcal{E}$,*
- *$\mathrm{lhs}(D) \subseteq T\sigma$ and $\mathrm{rhs}(D) \subseteq \mathsf{St}(T\sigma)$, and*
- *$T\sigma$ is well-typed in the environment $\mathcal{E}$.*

Proof. We proceed by induction on the length n of the derivation.

Base case ($n = 0$). In that case, $D = C$, $\mathrm{dom}(\sigma) = \emptyset$, and $T\sigma = T$, and we can thus trivially conclude.

Inductive case ($n \geq 1$). In that case, there exists a constraint system E, a reduction rule R amongst the ones described in Figure 4 page 31, and two substitutions σ_1 and σ_2 such that

$$C \overset{\mathsf{R}}{\rightsquigarrow}_{\sigma_1} E \rightsquigarrow_{\sigma_2}^{n-1} D \qquad \text{and} \qquad \sigma = \sigma_1\sigma_2$$

We first show that σ_1 is well-typed in the environment $\mathcal{E}$, $\mathrm{lhs}(E) \subseteq T\sigma_1$ and $\mathrm{rhs}(E) \subseteq \mathsf{St}(T\sigma_1)$, and $T\sigma_1$ is well-typed in the environment $\mathcal{E}$. We proceed by case analysis on the rule R.
Case $\mathsf{R} = \mathsf{R}_1$. In that case, $C = C' \wedge U \Vdash u$, $E = C'$ and $\mathrm{dom}(\sigma_1) = \emptyset$ for some constraint system C', some set of terms U and some term u.

- σ_1, the empty substitution, is trivially well-typed in the environment $\mathcal{E}$.

- $\mathrm{lhs}(E) = \mathrm{lhs}(C') \subseteq \mathrm{lhs}(C) \overset{\text{Hyp.}}{\subseteq} T = T\sigma_1$ and $\mathrm{rhs}(E) = \mathrm{rhs}(C') \subseteq \mathrm{rhs}(C) \overset{\text{Hyp.}}{\subseteq} \mathsf{St}(T) = \mathsf{St}(T\sigma_1)$
- $T\sigma_1 = T$ is by hypothesis well-typed in the environment $\mathcal{E}$.

Case $\mathsf{R} = \mathsf{R}_2$. In that case, $C = C' \wedge U \Vdash u$, $E = C\sigma_1$ and $\sigma_1 = \mathrm{mgu}(u, v)$ for some constraint system C', some set of terms U, and some terms $v \in \mathsf{St}(U)$ and u that are neither equal, nor variables, nor pairs.

- Since u and v are different, neither variables nor pairs but still unifiable, and since $\mathcal{Z}(T) = \emptyset$, it is necessarily the case that $u \in \mathsf{CryptSt}(\mathrm{rhs}(C)) \overset{\text{Hyp.}}{\subseteq} \mathsf{CryptSt}(\mathsf{St}(T)) = \mathsf{CryptSt}(T)$ and $v \in \mathsf{CryptSt}(U) \subseteq \mathsf{CryptSt}(\mathrm{lhs}(C)) \overset{\text{Hyp.}}{\subseteq} \mathsf{CryptSt}(T)$. Now because u and v are unifiable and T is well-typed in the environment $\mathcal{E}$, we can apply Proposition 2 page 38 and conclude that σ_1 is well-typed in the environment $\mathcal{E}$.
- $\mathrm{lhs}(E) = \mathrm{lhs}(C\sigma_1) = \mathrm{lhs}(C)\sigma_1 \overset{\text{Hyp.}}{\subseteq} T\sigma_1$ and $\mathrm{rhs}(E) = \mathrm{rhs}(C\sigma_1) = \mathrm{rhs}(C)\sigma_1 \overset{\text{Hyp.}}{\subseteq} \mathsf{St}(T)\sigma_1 \subseteq \mathsf{St}(T\sigma_1)$
- Because $u, v \in \mathsf{CryptSt}(T)$ (see first item of this case) are unifiable, and because T is by hypothesis well-typed in the environment $\mathcal{E}$, we know that u and v are of the same type in the environment $\mathcal{E}$. Hence T, u, and v satisfy the conditions of Proposition 3 page 39, which allows us to conclude that $T\sigma_1$ is well-typed in the environment $\mathcal{E}$.

Case $\mathsf{R} = \mathsf{R}_3$. In that case, $C = C' \wedge U \Vdash u$, $E = C\sigma_1$ and $\sigma_1 = \mathrm{mgu}(v, w)$ for some constraint system C', some set of terms U, some terms $v, w \in \mathsf{St}(U)$ that are neither equal, nor variables, nor pairs.

- As $v, w \in \mathsf{CryptSt}(U) \subseteq \mathsf{CryptSt}(\mathrm{lhs}(C)) \overset{\text{Hyp.}}{\subseteq} \mathsf{CryptSt}(T)$, this can be handled in a similar way as the first item of the previous case.
- Similar to the second item of the previous case.
- Similar to the third item of the previous case.

Case $\mathsf{R} = \mathsf{R}_4$. In that case, $C = C' \wedge U \Vdash u$, $E = C\sigma_1$ and $\sigma_1 = \mathrm{mgu}(v_2, v_3)$ for some constraint system C', some set of terms U, some term u, some term $\mathsf{aenc}(v_1, v_2) \in \mathsf{St}(U) \subseteq \mathsf{St}(T)$ such that $\mathsf{pv}(v_3) \in (\mathit{plaintext}(U) \cup \{\mathsf{pv}(\epsilon)\})$ and $v_2 \neq v_3$.

- Since T is by hypothesis well-typed, we know that $\mathcal{E} \triangleright v_2 : \alpha$. Moreover, the grammar of terms enforces $v_3 \in \mathcal{A} \cup \mathcal{Z}$ and thus according to our type system $\mathcal{E} \triangleright v_3 : \alpha$. Hence v_2 and v_3 are of the same type in the environment $\mathcal{E}$. Thus according to Proposition 2 page 38, σ_1 is well-typed in the environment $\mathcal{E}$.

- $\mathrm{lhs}(E) = \mathrm{lhs}(C\sigma_1) = \mathrm{lhs}(C)\sigma_1 \overset{\text{Hyp.}}{\subseteq} T\sigma_1$ and $\mathrm{rhs}(E) = \mathrm{rhs}(C\sigma_1) = \mathrm{rhs}(C)\sigma_1 \overset{\text{Hyp.}}{\subseteq} \mathsf{St}(T)\sigma_1 \subseteq \mathsf{St}(T\sigma_1)$
- As we just saw in the first item of this case, $\mathcal{E} \triangleright v_2 : \alpha$ and $\mathcal{E} \triangleright v_3 : \alpha$. Thus $\sigma_1 = \{v_2 \mapsto v_3\}$ or $\sigma_1 = \{v_3 \mapsto v_2\}$ is well-typed, and $T\sigma_1$ is trivially well-typed in the environment $\mathcal{E}$ (since σ_1 just replaces α typed variables with α typed terms).

Case $\mathsf{R} = \mathsf{R}_5$. This case cannot occur, because it would contradict the hypothesis according to which D admits a solution.
Case $\mathsf{R} = \mathsf{R}_f$. In that case, $C = C' \wedge U \Vdash f(u_1, \ldots, u_m)$, $E = C' \wedge \bigwedge_{i \in [\![m]\!]} U \Vdash u_i$ and $\mathrm{dom}(\sigma_1) = \emptyset$ for some constraint system C', some set of terms U, some terms $u_1, \ldots, u_m$ and some function symbol f.

- σ_1 is trivially well-typed in the environment $\mathcal{E}$.
- $\mathrm{lhs}(E) = \mathrm{lhs}(C) \overset{\text{Hyp.}}{\subseteq} T = T\sigma_1$ and because $\{u_1, \ldots, u_m\} \subseteq \mathsf{St}(f(u_1, \ldots, u_m)) \subseteq \mathsf{St}(\mathrm{rhs}(C)) \overset{\text{Hyp}}{\subseteq} \mathsf{St}(T)$ we also have $\mathrm{rhs}(E) = \mathrm{rhs}(C') \cup \{u_1, \ldots, u_m\} \subseteq \mathrm{rhs}(C) \cup \{u_1, \ldots, u_m\} \overset{\text{Hyp.}}{\subseteq} \mathsf{St}(T) = \mathsf{St}(T\sigma_1)$
- $T\sigma_1 = T$ is by hypothesis well-typed in the environment $\mathcal{E}$.

At this point we have established that $\mathcal{E}$, $T\sigma_1$, E, D, and σ_2 satisfy the conditions for applying our inductive hypothesis, and we can thus conclude by induction that

- σ_2 is well-typed in the environment $\mathcal{E}$,
- $\mathrm{lhs}(D) \subseteq (T\sigma_1)\sigma_2$ and $\mathrm{rhs}(D) \subseteq \mathsf{St}((T\sigma_1)\sigma_2)$, and
- $(T\sigma_1)\sigma_2$ is well-typed in the environment $\mathcal{E}$.

which in turns imply that

- $\sigma = \sigma_1\sigma_2$ is well-typed in the environment $\mathcal{E}$ since σ_1 and σ_2 are well-typed in the environment $\mathcal{E}$, and have disjoint domains,
- $\mathrm{lhs}(D) \subseteq T\sigma = (T\sigma_1)\sigma_2$ and $\mathrm{rhs}(D) \subseteq \mathsf{St}(T\sigma) = \mathsf{St}((T\sigma_1)\sigma_2)$, and
- $T\sigma = (T\sigma_1)\sigma_2$ is well-typed in the environment $\mathcal{E}$.

and terminates our proof by induction. □

6.3. Proof of the main result

We can now proceed with the proof of our main result, namely Theorem 1, restated below.

Theorem 1. *Let* $\Pi \in \mathbb{P}$ *be a protocol,* T_0 *a set of ground atomic terms and* $\phi \in \Phi(\Pi)$ *a security property. If there exists a symbolic trace* tr *of* Π *and a ground substitution* σ *such that:*

1. $\mathsf{tr}\sigma$ *is a valid execution of* Π w.r.t. *the initial intruder knowledge* T_0*, and*
2. $\langle \mathsf{tr}\sigma, T_0 \rangle \models \neg\phi$*,*

then there also exists a ground substitution σ_{wt} *such that:*

1. $\mathsf{tr}\sigma_{\mathsf{wt}}$ *is a valid execution of* Π w.r.t. *the initial intruder knowledge* T_0*,*
2. $\mathsf{tr}\sigma_{\mathsf{wt}}$ *is well-typed in the environment* $\mathcal{E} = \langle \Pi, \phi \rangle$*, and*
3. $\langle \mathsf{tr}\sigma_{\mathsf{wt}}, T_0 \rangle \models \neg\phi$

Proof. Let Π be a protocol in $\mathbb{P}$, T_0 a set of ground atomic terms, ϕ a property in $\Phi(\Pi)$. Let also tr be a symbolic trace of Π, and σ a substitution such that

1. $\mathsf{tr}\sigma$ is a valid execution of Π *w.r.t.* the initial intruder knowledge T_0, and
2. $\langle \mathsf{tr}\sigma, T_0 \rangle \models \neg\phi$,

In other words, let $\mathsf{tr}\sigma$ be a valid execution of Π, *w.r.t.* the initial intruder knowledge T_0, that violates the property ϕ. Let $\mathcal{E} = \langle \Pi, \phi \rangle$ be the corresponding typing environment. We will build the substitution σ_{wt}, well-typed in the environment $\mathcal{E}$, such that $\mathsf{tr}\sigma_{\mathsf{wt}}$ is a well-typed valid execution of Π, *w.r.t.* the initial intruder knowledge T_0, that also violates the property ϕ.

Let $\xi = \mathbf{T}(\neg\phi, \mathsf{tr}, T_0)$ be the elementary formula corresponding to $\neg\phi$. According to Lemma 7 page 35 we know that $\langle \mathsf{tr}\sigma, T_0 \rangle \models \neg\phi$ if and only if $\sigma \models' \xi$. Moreover, we know that ξ is of the form $\xi = \exists x_1, \dots \exists x_k,\ \xi'$, where ξ' is a quantifier free elementary formula that can be rewritten, as said in Section 6.1.4, into $\psi = \bigvee_{i \in [\![n]\!]} C_i \wedge Eq_i \wedge Deq_i$. Hence $\sigma \models' \exists x_1, \dots \exists x_k,\ \xi'$ if and only if $\sigma \models' \exists x_1, \dots \exists x_k,\ \psi$, and accordingly $\langle \mathsf{tr}\sigma, T_0 \rangle \models \neg\phi$ if and only if $\sigma \models' \exists x_1, \dots \exists x_k,\ \psi$.

Let $C = \mathcal{C}str(\mathsf{tr}, T_0)$ be the constraint system corresponding to the symbolic trace tr and the initial intruder knowledge T_0 such as specified in Definition 23 page 30. Since $\mathsf{tr}\sigma$ is a valid execution, the constraint system C admits (at least) a solution D, and thus there exists a substitution σ_1, that can be computed using simplification rules of Figure 4 page 31, such that $C \rightsquigarrow^{\ell}_{\sigma_1} D$. Now D is a constraint system in solved form and we can apply procedure $\mathbf{D}$ to (D, ψ). According to Lemma 4 page 37 the procedure $\mathbf{D}$ is sound and complete, and given that $\sigma \models' \exists x_1, \dots \exists x_k,\ \psi$, the procedure will find an attack. We deduce that there exists an $i \in [\![n]\!]$ (let i_{attack} be such an i) such that the procedure stops at step i finding an attack. We then get two substitutions σ_2 and σ_3, and a constraint system E in solved form such that

- $(D\sigma_2 \wedge C_{i_{\mathsf{attack}}}\sigma_1\sigma_2) \rightsquigarrow^h_{\sigma_3} E$
- $\sigma_1\sigma_2\sigma_3\sigma_F \models' \exists x_1, \ldots \exists x_k,\ \psi$

with $\sigma_2 = \mathrm{mgu}((Eq_{i_{\mathsf{attack}}})\sigma_1)$. Again, thanks to Lemma 7 page 35 and to the fact that ξ' and ψ are equivalent, we can deduce that $\sigma_1\sigma_2\sigma_3\sigma_F \models' \neg\phi$. Our first step is to establish that $\sigma'_{\mathsf{wt}} = \sigma_1\sigma_2\sigma_3$ is well-typed. The domains of substitutions σ_1, σ_2 and σ_3 being disjoint and each of these substitutions being applied to $C \wedge C_{i_{\mathsf{attack}}}$, $Eq_{i_{\mathsf{attack}}}$ and $Deq_{i_{\mathsf{attack}}}$ as soon as computed, it suffices to separately establish that each of these substitutions σ_1, σ_2 and σ_3 is well-typed.

Then we will define a substitution σ''_{wt} that will be well-typed and play the same role as σ_F *i.e.* substituting uninstantiated variables while preserving disequalities.

Let

$$T = \mathrm{lhs}(C) \cup \mathrm{rhs}(C) \ \cup \bigcup_{t=u \in Eq_{i_{\mathsf{attack}}}} \{t, u\} \ \cup \ \mathrm{lhs}(C_{i_{\mathsf{attack}}}) \cup \mathrm{rhs}(C_{i_{\mathsf{attack}}})$$

By construction of C we know that

$$\mathsf{CryptSt}(\mathrm{lhs}(C) \cup \mathrm{rhs}(C)) \subseteq \mathsf{CryptSt}(\mathsf{tr})$$

By definition of the transformation $\mathbf{T}$, and because the set of considered formulas only have atomic subterms (Condition 1 of Definition 20 page 26),

$$\mathsf{CryptSt}(\mathrm{lhs}(C_{i_{\mathsf{attack}}})) \subseteq \mathsf{CryptSt}(\mathsf{tr}) \quad \text{and} \quad \mathsf{CryptSt}(\mathrm{rhs}(C_{i_{\mathsf{attack}}})) \subseteq \mathsf{CryptSt}(\phi) \stackrel{\text{Hyp.}}{=} \emptyset$$

and by definition of $\mathbf{T}$ we know that for all $t = u \in Eq_{i_{\mathsf{attack}}}$

$$\mathsf{CryptSt}(t) \subseteq \mathsf{CryptSt}(\mathsf{tr}) \quad \text{and} \quad \mathsf{CryptSt}(u) \subseteq \mathsf{CryptSt}(\phi) \stackrel{\text{Hyp.}}{=} \emptyset$$

By combining all these with $\mathsf{St}(\phi) \subseteq \mathcal{A} \cup \mathcal{X}$ we obtain

$$\mathsf{CryptSt}(T) \subseteq \mathsf{CryptSt}(\mathsf{tr})$$

Let $u, v \in \mathsf{CryptSt}(T)$ such that $\mathrm{mgu}(u, v) \neq \bot$. Condition 3 of Definition 19 page 24 implies according to Proposition 1 page 24 (see Condition 3') that there exists τ such that $\mathcal{E} \triangleright u : \tau$ and $\mathcal{E} \triangleright v : \tau$. Moreover, for $\mathsf{aenc}(u, v) \in \mathsf{CryptSt}(T)$, Condition 2 of Definition 19 implies according to Proposition 1 (see Condition 2'), that $\mathcal{E} \triangleright v : \alpha$. Thus T is well-typed in the environment $\mathcal{E}$.

σ_1 and $T\sigma_1$ are well-typed.
C admits by hypothesis a solution (the constraint system D for example) and by construction $\mathrm{lhs}(C) \subseteq T$ and $\mathrm{rhs}(C) \subseteq T \subseteq \mathsf{St}(T)$. Moreover, we have just established that T is well-typed. Thus $\mathcal{E}$, T, C, D and σ_1 satisfy the hypotheses of Proposition 4 page 40 that allows us to conclude that σ_1 and $T\sigma_1$ are well-typed, and that $\mathrm{lhs}(D) \subseteq T\sigma_1$ and $\mathrm{rhs}(D) \subseteq \mathsf{St}(T\sigma_1)$.

σ_2 and $(T\sigma_1)\sigma_2$ are well-typed.
Remember that $\sigma_2 = \text{mgu}((Eq_{i_{\mathsf{attack}}})\sigma_1)$ and let $(Eq_{i_{\mathsf{attack}}})\sigma_1 = \{t_j = u_j\}_{j\in[\![r]\!]}$. Thus $\sigma_2 = \theta_1 \dots \theta_r$ for

$$\theta_j = \text{mgu}(t_j\theta_1 \dots \theta_{j-1}, u_j\theta_1 \dots \theta_{j-1})$$

for all $j \in [\![r]\!]$, $t_j, u_j \in T\sigma_1$. We prove by induction on r that

- σ_2 is well-typed, and that
- $(T\sigma_1)\sigma_2$ is well-typed.

Base case ($r = 0$). In that case, $\text{dom}(\sigma_2) = \emptyset$ is thus trivially well-typed. Moreover, since $T\sigma_1$ is well-typed, so is $T\sigma_1\sigma_2 = T\sigma_1$.

Inductive case ($r \geq 1$). In that case, we know by inductive hypothesis that $\theta_1 \dots \theta_{r-1}$ and $T\sigma_1\theta_1 \dots \theta_{r-1}$ are well-typed. Moreover, $t_r = u_r \in Eq_{i_{\mathsf{attack}}}\sigma_1$ implies that there exists $t'_r = u'_r \in Eq_{i_{\mathsf{attack}}}$ such that $t_r = t'_r\sigma_1$ and $u_r = u'_r\sigma_1$. But because of Condition 2 of Definition 20 page 26 of the class $\Phi(\Pi)$, it follows that t'_r and u'_r are of the same type, *i.e.* $\exists\tau$, $\mathcal{E}\triangleright t'_r : \tau \wedge \mathcal{E}\triangleright u'_r : \tau$. And since σ_1 and $(\theta_1 \dots \theta_{r-1})$ are two well-typed substitutions we deduce that $t'_r\sigma_1(\theta_1 \dots \theta_{r-1})$ and $u'_r\sigma_1(\theta_1 \dots \theta_{r-1})$ are also of the same type, *i.e.* $\mathcal{E} \triangleright t'_r\sigma_1\theta_1 \dots \theta_{r-1} : \tau$ and $\mathcal{E} \triangleright u'_r\sigma_1\theta_1 \dots \theta_{r-1} : \tau$. According to Proposition 2 page 38, the substitution

$$\theta_r = \text{mgu}(t'_r\sigma_1\theta_1 \dots \theta_{r-1}, u'_r\sigma_1\theta_1 \dots \theta_{r-1}) = \text{mgu}(t_r\theta_1 \dots \theta_{r-1}, u_r\theta_1 \dots \theta_{r-1})$$

is thus well-typed. Its domain being disjoint of the domain of $\theta_1 \dots \theta_{r-1}$ we can conclude that $\sigma_2 = \theta_1 \dots \theta_{r-1}\theta_r$ is well-typed. Finally, note that $t_r, u_r \in T\sigma_1$ implies that $t_r\theta_1 \dots \theta_{r-1}, u_r\theta_1 \dots \theta_{r-1} \in T\sigma_1\theta_1 \dots \theta_{r-1}$. Hence, all the conditions of Proposition 3 page 39 hold, which allows us to deduce that $T\sigma_1\theta_1 \dots \theta_r = T\sigma_1\sigma_2$ is well-typed and conclude our proof by induction.

We have thus showed that σ_2 and $T\sigma_1\sigma_2$ are well-typed in the environment $\mathcal{E}$.

σ_3 is well-typed.
Knowing that $\text{lhs}(D) \subseteq T\sigma_1$ and $\text{rhs}(D) \subseteq \mathsf{St}(T\sigma_1)$ (see above in the proof), we can deduce that $\text{lhs}(D\sigma_2) = \text{lhs}(D)\sigma_2 \subseteq (T\sigma_1)\sigma_2$ and $\text{rhs}(D)\sigma_2 = \text{rhs}(D\sigma_2) \subseteq (\mathsf{St}(T\sigma_1))\sigma_2 \subseteq \mathsf{St}((T\sigma_1)\sigma_2)$. In the same way, knowing $\text{lhs}(C_{i_{\mathsf{attack}}}) \subseteq T$ and that $\text{rhs}(C_{i_{\mathsf{attack}}}) \subseteq T$ (by construction of T), we can deduce that $\text{lhs}((C_{i_{\mathsf{attack}}}\sigma_1)\sigma_2) \subseteq (T\sigma_1)\sigma_2$ and $\text{rhs}((C_{i_{\mathsf{attack}}}\sigma_1)\sigma_2) \subseteq ((\mathsf{St}(T))\sigma_1)\sigma_2 \subseteq \mathsf{St}((T\sigma_1)\sigma_2)$. Thus

$$\text{lhs}(D\sigma_2 \wedge (C_{i_{\mathsf{attack}}}\sigma_1)\sigma_2) = \text{lhs}(D\sigma_2) \cup \text{lhs}((C_{i_{\mathsf{attack}}}\sigma_1)\sigma_2) \subseteq T\sigma_1\sigma_2$$

and

$$\text{rhs}(D\sigma_2 \wedge (C_{i_{\mathsf{attack}}}\sigma_1)\sigma_2) = \text{rhs}(D\sigma_2) \cup \text{rhs}((C_{i_{\mathsf{attack}}}\sigma_1)\sigma_2) \subseteq \mathsf{St}((T\sigma_1)\sigma_2)$$

$T\sigma_1\sigma_2$ and $(D\sigma_2 \wedge C_{i_{\mathsf{attack}}}\sigma_1\sigma_2)$ thus satisfy the hypotheses of Proposition 4 page 40 which allows to conclude that σ_3 is well-typed.

At this stage we have established that $\sigma'_{\mathsf{wt}} = \sigma_1\sigma_2\sigma_3$ is well-typed. We are thus left with defining the well-typed substitution σ''_{wt} such that its domain includes all the variables of tr not in the domain of σ'_{wt}, *i.e.* the right members of E. The role of σ''_{wt} is the same as the role of σ_F in Procedure 1 page 36, *i.e.* to instantiate all the variables not in the domain of σ'_{wt} so as to respect inequalities in $Deq_{i_{\mathsf{attack}}}$. More precisely, σ''_{wt} will map each variable of its domain to an unused term of the same type. Let's remind that the only constraints concerning the construction of σ''_{wt} are the following:

- for all variables $x \in \mathrm{dom}(\sigma''_{\mathsf{wt}})$, $U \vdash \sigma''_{\mathsf{wt}}(x)$, where $U \Vdash x \in E$
- $t\sigma''_{\mathsf{wt}} \neq u\sigma''_{\mathsf{wt}}$ for all $t \neq u \in (Deq_{i_{\mathsf{attack}}})\sigma'_{\mathsf{wt}}$

Now we need to explain how to build the well-typed substitution σ''_{wt}. The purpose of this substitution is to replace all the variables remaining in tr after application of σ'_{wt}. Since these variables have not been substituted, it means that their actual ground value is irrelevant for the attack. We just need to substitute them in such a way that σ''_{wt} is well-typed and they preserve the truth of inequalities. The substitution σ''_{wt} is then defined as follows: $\mathrm{dom}(\sigma''_{\mathsf{wt}}) = \mathrm{rhs}(E)$, and for all $x \in \mathrm{dom}(\sigma''_{\mathsf{wt}})$, $\sigma''_{\mathsf{wt}}(x) = \mathsf{fake}(\tau)$ where τ is the type of x in the environment $\mathcal{E} = \langle \Pi, \phi \rangle$, *i.e.* $\mathcal{E} \triangleright x : \tau$. The function $\mathsf{fake}()$, when applied to a type τ, returns a term t such that $\mathcal{E} \triangleright t : \tau$. This function is defined inductively as follows:

$$
\begin{array}{ll}
\mathsf{fake}(\alpha) = a & \text{with } a \in \mathcal{A} \text{ and } a \text{ fresh} \\
\mathsf{fake}(\gamma_c) = c & \text{if } c \in \mathcal{C} \\
\mathsf{fake}(\nu_n) = n^{\epsilon,k} & \text{if } n \in \mathcal{N}(\Pi) \text{ and } k \in \mathbb{N} \text{ and } n^{\epsilon,k} \text{ fresh} \\
\mathsf{fake}(\mathsf{pv}(\alpha)) = \mathsf{pv}(\epsilon) & \\
\mathsf{fake}(\mathsf{sh}(\alpha,\alpha)) = \mathsf{sh}(\epsilon, a) & \text{with } a \in \mathcal{A} \text{ and } a \text{ fresh} \\
\mathsf{fake}(f(\tau_1,\ldots,\tau_n)) = f(\mathsf{fake}(\tau_1),\ldots,\mathsf{fake}(\tau_n)) & \text{for } f \in \mathcal{F}
\end{array}
$$

One can easily see that for every type τ induced by Π, $\mathcal{N}_\epsilon(\Pi) \vdash \mathsf{fake}(\tau)$, and having provided the intruder with an unbounded number of nonces of each type, that σ''_{wt} is well-typed. Moreover, for all $t \neq u \in Deq_{i_{\mathsf{attack}}}\sigma'_{\mathsf{wt}}$, there exists $t' \neq u' \in Deq_{i_{\mathsf{attack}}}$ such that $t = t'\sigma'_{\mathsf{wt}}$ and $u = u'\sigma'_{\mathsf{wt}}$ with $t \in \mathsf{St}(\mathsf{tr})$ and $u \in \mathsf{St}(\mathsf{Sf}^+(\phi))$. According to the translation of formulas given in Definition 27 page 34, these disequalities can either come from a subformula $\mathsf{C}(t)$, in which case t is of type α, or from a status event occurring in the original formula. Now, according to Condition 3 of Definition 20 page 26, we know that t and u do not admit any subtype of constant's type nor private key's type. Thus, having selected agents that do not appear in u for $\mathsf{fake}(\alpha)$ and $\mathsf{fake}(\mathsf{sh}(\alpha,\alpha))$, and fresh nonces for $\mathsf{fake}(\nu)$, we know that the inequalities are satisfied.

Above we announced that $\sigma'_{\mathsf{wt}}\sigma''_{\mathsf{wt}} \models' \exists x_1, \ldots \exists x_k, \psi$ with $\sigma_{\mathsf{wt}} = \sigma'_{\mathsf{wt}}\sigma''_{\mathsf{wt}}$ well-typed. Thus $\langle \mathsf{tr}\sigma_{\mathsf{wt}}, T_0 \rangle \models \neg\phi$. Moreover by soundness of procedure **D** we know that $\mathsf{tr}\sigma_{\mathsf{wt}}$ is a valid execution of Π, *w.r.t.* the initial intruder knowledge T_0 which concludes our proof. □

7. Application to decidability results

The main result stated in Section 5.3 is rather formal but has a direct corollary that is more simple to state. It says that, for protocols in our class, we only need to consider bounded messages when looking for an attack.

Corollary 1. *If a protocol* $\Pi \in \mathbb{P}$ *admits an attack then it also admits an attack along which all messages have size at most as large as a message of* Π.

The proof of this result is rather simple and consists in two steps. First for every term t of type τ we have, by induction on τ, that $|t| \leq |\tau|$. Indeed, a variable is of size 1, but its type is the type of the honest substitution for this variable, which can be composed. For example a variable may expect to be instantiated by a pair of nonces, so while the variable is of size 1 its type is of size 3. Otherwise, the size of complex types depends only on the function symbol, similarly to the size of terms. Then, given the symbolic trace tr associated to the attack, we can use Theorem 1 page 27 and the existence of a well-typed substitution σ_{wt} to say that any subterm of $\mathsf{tr}\sigma_{\mathsf{wt}}$ has a type that is a subtype of one message of an honest execution of Π, which allows us to conclude.

7.1. Tagging

In this section we give a new class of protocols to which our main result can be applied. The main reasons to consider this new class $\mathbb{P}_{\mathrm{tag}}$ are its applicability in practice and the simplicity to check if a protocol belongs to the class. More precisely, we are going to see how to transform a protocol into a protocol in $\mathbb{P}$, while keeping its intended "semantics". This transformation consists in annotating each application of a cryptographic primitive as described in the following definition.

Definition 29 (The class of protocols $\mathbb{P}_{\mathbf{tag}}$). The class of protocols $\mathbb{P}_{\mathrm{tag}}$ is the set of protocols $\Pi = [e_1; \ldots; e_\ell]$ satisfying the following 4 conditions:

1. each predicate appears in at most one status event of the protocol, *i.e.* for all $i, j \in [\![\ell]\!]$, if $i \neq j$, $e_i = \mathsf{Q}(t_1, \ldots, t_n)$ and $e_j = \mathsf{Q}'(u_1, \ldots, u_m)$, then $\mathsf{Q} \neq \mathsf{Q}'$,
2. for all $\mathsf{aenc}(u, v) \in \mathsf{CryptSt}(\Pi)$, $\delta_\Pi(v) \in \mathcal{A}$,
3. for all $f(u_1, \ldots, u_n) \in \mathsf{CryptSt}(\Pi)$, there exists a constant $c \in \mathcal{C}$ and a term u'_1 such that $u_1 = \langle c, u'_1 \rangle$, and
4. for all $f(\langle c, u_1 \rangle, \ldots, u_n), f(\langle d, v_1 \rangle, \ldots, v_n) \in \mathsf{CryptSt}(\Pi)$

$$(f(\langle c, u_1 \rangle, \ldots, u_n))\delta_\Pi \neq (f(\langle d, v_1 \rangle, \ldots, v_n))\delta_\Pi \;\Rightarrow\; c \neq d.$$

The benefits for introducing this new class $\mathbb{P}_{\mathrm{tag}}$ in terms of usability and security will be explained below. However, it is also worth mentioning that this definition makes it easy to check whether a protocol fits in the class or not. There is no need to consider explicitly the typing system (thanks to Condition 1).

Condition 1 forces the protocol Π to be well-typed and thus implies Condition 1 of Definition 19 page 24. Condition 2 imposes each honest asymmetric encryption in Π to use the public key of an agent in $\mathcal{A}$, thus implying Condition 2 of Definition 19. Condition 3 requires each application of a cryptographic primitive to be annotated (or *statically tagged*) with a constant c, while Condition 4 implies that two encrypted subterms that do not correspond to the same message of the protocol are tagged with different constants. The idea behind these tags is to restrict the unifiability of subterms in Π in the sense of Condition 3 of Definition 19.

Indeed, let t and u be two encrypted subterms of Π but statically tagged with different constants; t and u are then non-unifiable and of different type. Moreover, let sc be a scenario of Π with the corresponding symbolic trace tr, if $t' \in \mathsf{CryptSt}(\mathsf{tr})$ is an instance of t and $u' \in \mathsf{CryptSt}(\mathsf{tr})$ is an instance of u, then t' and u' are also not unifiable and of different types. Conversely, let v and w be two encrypted subterms of trace tr, and let v' be the encrypted subterm of Π that v instantiates, and w' the encrypted subterm of Π that w instantiates. If v and w are unifiable then they are necessarily tagged with the same constant (otherwise the unification would fail). Condition 4 of Definition 29 thus ensures that v' and w' correspond to the same message in the honest execution of the protocol, and that they are thus of the same type. But then v and w are also of the same type. We recover in this way the non-unifiability of subterms of different types. Conditions 3 and 4 of Definition 29 combined imply Condition 3 of Definition 19.

The interest of this fragment of the class $\mathbb{P}$ is twofold. The first is that it is possible to transform a protocol satisfying Conditions 1 and 2 of Definition 19 into a protocol in $\mathbb{P}_{\mathrm{tag}}$, just by annotating each application of a cryptographic primitive by a constant. Now, many protocols of interest satisfy Condition 2 of Definition 29. As far as Condition 1 of Definition 29 is concerned, let's remember that status events are added to a protocol in order to specify the security property of interest. Status events introduced in the specification of protocol thus depend on the considered formula. For the usual properties, like the ones presented in Section 4.2 the introduced status events are all different. Hence, if we focus on these properties, it is possible to transform many existing protocols into protocols of $\mathbb{P}_{\mathrm{tag}}$. Moreover, the cost of this transformation is very low. Indeed, the static tagging only increases the size of a message with a few bits. It increases neither the number of exchanged messages, nor the number of cryptographic computations.

The second interest lies in the fact that tagged protocols are at least as secure as their untagged counterparts. Many known attacks rely on the fact that an agent will decrypt a message thinking he is decrypting a message of different type. These attacks thus rest on confusions about the types of messages exchanged during the execution of the protocol. As our example below shows, tagged protocols can be more secure than their untagged counterparts, precisely because the type of each message is made explicit through the tag. As a matter of fact, this is one of the prescribed techniques by M. Abadi and R. Needham in [AN96] for the design of "secure" protocols.

Example 22. *We illustrate our comments by tagging our running example protocol* Π_{Toy}

$$\begin{array}{rl}\widehat{\Pi_{\mathsf{Toy}}} = [& \mathsf{snd}(z_a, z_b, \mathsf{aenc}(\langle 1, \langle \mathsf{aenc}(\langle 2, n\rangle, z_b), z_a\rangle\rangle, z_b));\\ & \mathsf{Secret}(z_a, z_a, z_b, n);\\ & \mathsf{rcv}(z_b, z_a, \mathsf{aenc}(\langle 1, \langle \mathsf{aenc}(\langle 2, x\rangle, z_b), z_a\rangle\rangle, z_b));\\ & \mathsf{snd}(z_b, z_a, \mathsf{aenc}(\langle 3, \langle \mathsf{aenc}(\langle 4, x\rangle, z_a), z_b\rangle\rangle, z_a));\\ & \mathsf{rcv}(z_a, z_b, \mathsf{aenc}(\langle 3, \langle \mathsf{aenc}(\langle 4, n\rangle, z_a), z_b\rangle\rangle, z_a))]\end{array}$$

Note, that tagged as such the encrypted subterms

$$\mathsf{aenc}(\langle 1, \langle \mathsf{aenc}(\langle 2, n\rangle, b), a\rangle\rangle, b)) \text{ and } \mathsf{aenc}(\langle 2, x\rangle, b)$$

are not unifiable anymore, and that the attack on secrecy presented in Example 7 page 13 cannot be mounted against $\widehat{\Pi_{\mathsf{Toy}}}$*. In fact, this tagged version of* Π_{Toy} *ensures secrecy of the nonce* n*.*

7.2. Discussion on (un)decidability

The class of protocols $\mathbb{P}$, and in particular its fragment $\mathbb{P}_{\mathrm{tag}}$, has been studied in the past. With extra restrictions some decidability results have been obtained. B. Blanchet and A. Podelski in [BP05] showed that if only a bounded number of fresh nonces is considered then secrecy and authentication are decidable. On the other hand R. Ramanujam and S. P. Suresh [RS03b] as well as G. Lowe in [Low99, HLS03] showed that under some extra conditions secrecy is decidable for the class $\mathbb{P}_{\mathrm{tag}}$. These restrictions exclude protocols relying on temporary secrets or blind copies (*i.e.* variables of non-atomic type).

However, reintroducing temporary secrets dramatically complicates the problem. Indeed, it is sufficient to consider one session of each role to find an attack on secrecy if such temporary secrets are forbidden (see [RS03b, Low99, HLS03]). However, we will now see that for any $k \in \mathbb{N}$, it is possible to build a protocol Π_k in $\mathbb{P}_{\mathrm{tag}}$ that admits an attack on the secrecy property ϕ_k that requires $2k+1$ sessions to be mounted.

For $k = 1$, Π_1 corresponds to the following sequence of events:

$$\begin{array}{rll}\Pi_1 = [& \mathsf{snd}(a, b, & \langle \mathsf{aenc}(\langle 1, \langle a, \langle n_1, n_2\rangle\rangle\rangle, b), \mathsf{sign}(\langle 2, \mathsf{aenc}(\langle 3, \langle a, n_2\rangle\rangle, b)\rangle, a)\rangle);\\ & \mathsf{rcv}(b, a, & \langle \mathsf{aenc}(\langle 1, \langle a, \langle x_1, x_2\rangle\rangle\rangle, b), \mathsf{sign}(\langle 2, \mathsf{aenc}(\langle 3, \langle a, x_2\rangle\rangle, b)\rangle, a)\rangle);\\ & \mathsf{snd}(b, a, & \langle x_2, \mathsf{senc}(\langle 4, n_3\rangle, x_1))\rangle);\\ & \mathsf{Secret}_1(b, a, b, n_3); & \\ & \mathsf{rcv}(a, b, & \langle n_2, \mathsf{senc}(\langle 4, x_3\rangle, n_1))\rangle)]\end{array}$$

and the secrecy property can be specified using the following formula:

$$\phi_1 = \forall y.\ \forall y_a.\ \forall y_b.\ \forall y_n.\ \Diamond(\mathsf{Secret}_1(y, y_a, y_b, y_n)) \ \Rightarrow\ \neg\mathsf{learn}(y_n)$$

The following execution exec_1 is a valid execution of Π_1 *w.r.t.* any initial intruder knowledge T_0, that violates the secrecy property ϕ_1. For the sake of

clarity, nonces known by the intruder are underlined once, and nonces generated by the intruder are underlined twice.

$$\begin{array}{lll}
\mathsf{exec}_1 = [& \mathsf{snd}(a,b, & \langle \mathsf{aenc}(\langle 1, \langle a, \langle n_1^1, n_2^1\rangle\rangle\rangle, b), \mathsf{sign}(\langle 2, \mathsf{aenc}(\langle 3, \langle a, n_2^1\rangle\rangle, b)\rangle, a)\rangle); \\
& \mathsf{rcv}(b,a, & \langle \mathsf{aenc}(\langle 1, \langle a, \langle n_1^1, n_2^1\rangle\rangle\rangle, b), \mathsf{sign}(\langle 2, \mathsf{aenc}(\langle 3, \langle a, n_2^1\rangle\rangle, b)\rangle, a)\rangle); \\
& \mathsf{snd}(b,a, & \langle \underline{n_2^1}, \mathsf{senc}(\langle 4, n_3^2\rangle, n_1^1)\rangle); \\
& \mathsf{rcv}(b,a, & \langle \mathsf{aenc}(\langle 1, \langle a, \langle \underline{n_1^\epsilon}, \underline{n_2^1}\rangle\rangle\rangle, b), \mathsf{sign}(\langle 2, \mathsf{aenc}(\langle 3, \langle a, \underline{n_2^1}\rangle\rangle, b)\rangle, a)\rangle); \\
& \mathsf{snd}(b,a, & \langle n_2^1, \mathsf{senc}(\langle 4, \underline{\underline{n_3^3}}\rangle, \underline{\underline{n_1^\epsilon}})\rangle); \\
& \mathsf{Secret}_1(a,b,n_3^3)] &
\end{array}$$

By learning the temporary secret n_2^1, the intruder can introduce his own nonce n_1^ϵ in key position in the message containing the secret n_3^3. This attack requires 3 sessions.

This construction can be generalised to any k inductively as follows. Suppose that k is even. Let Π_{k-1} be the protocol obtained accordingly and such that $\mathsf{Secret}_{k-1}(b,a,b,n) \in \mathsf{Evts}(\Pi_{k-1})$ and $x \in \mathcal{X}(\Pi_{k-1})$ such that $\delta_{\Pi_{k-1}}(x) = n$. Let also $c_1, c_2, c_3, c_4 \in (\mathcal{C} \setminus \mathcal{C}(\Pi_{k-1}))$ be four new constants, $m_1, m_2 \in (\mathcal{N} \setminus \mathcal{N}(\Pi_{k-1}))$ two new nonces, and $y_1, y_2 \in (\mathcal{X} \setminus \mathcal{X}(\Pi_{k-1}))$ two new variables. We define the protocol $\Pi_k = \Pi'_k \mathbin{@} \Pi_{k-1}$ where

$$\begin{array}{lll}
\Pi'_k = [& \mathsf{snd}(b,a, & \langle \mathsf{aenc}(\langle c_1, \langle b, \langle m_1, n\rangle\rangle\rangle, a), \mathsf{sign}(\langle c_2, \mathsf{aenc}(\langle c_3, \langle b, n\rangle\rangle, a)\rangle, b)\rangle); \\
& \mathsf{rcv}(a,b, & \langle \mathsf{aenc}(\langle c_1, \langle b, \langle y_1, x\rangle\rangle\rangle, a), \mathsf{sign}(\langle c_2, \mathsf{aenc}(\langle c_3, \langle b, x\rangle\rangle, a)\rangle, b)\rangle); \\
& \mathsf{snd}(a,b, & \mathsf{senc}(\langle c_4, m_2\rangle, y_1)); \\
& \mathsf{Secret}_k(a,a,b,m_2); & \\
& \mathsf{rcv}(b,a, & \mathsf{senc}(\langle c_4, y_2\rangle, m_1))]
\end{array}$$

And we consider the property

$$\phi_k = \forall y.\ \forall y_a.\ \forall y_b.\ \forall y_n.\ \Diamond(\mathsf{Secret}_k(y, y_a, y_b, y_n)) \ \Rightarrow\ \neg\mathsf{learn}(y_n)$$

To attack Π_k the intruder will first have to obtain the secret n of Π_{k-1} (n is the analogous to n_2^1 in Π_1 temporary secret), and then mimic the attack against Π_1. The case where k is odd is analogous except that the secret of Π_{k-1} is generated by a and that we thus need to invert in the constructions the names of a and b.

To illustrate this construction let's build the protocol Π_2 from protocol Π_1.

$$\begin{array}{lll}
\Pi_2 = [& \mathsf{snd}(b,a, & \langle \mathsf{aenc}(\langle 5, \langle b, \langle n_4, n_3\rangle\rangle\rangle, a), \mathsf{sign}(\langle 6, \mathsf{aenc}(\langle 7, \langle b, n_3\rangle\rangle, a)\rangle, b)\rangle); \\
& \mathsf{rcv}(a,b, & \langle \mathsf{aenc}(\langle 5, \langle b, \langle y_4, x_3\rangle\rangle\rangle, a), \mathsf{sign}(\langle 6, \mathsf{aenc}(\langle 7, \langle b, x_3\rangle\rangle, a)\rangle, b)\rangle); \\
& \mathsf{snd}(a,b, & \mathsf{senc}(\langle 8, n_5\rangle, y_4)); \\
& \mathsf{Secret}_2(a,a,b,n_5); & \\
& \mathsf{rcv}(b,a, & \mathsf{senc}(\langle 8, y_5\rangle, n_4)); \\
 & & \\
& \mathsf{snd}(a,b, & \langle \mathsf{aenc}(\langle 1, \langle a, \langle n_1, n_2\rangle\rangle\rangle, b), \mathsf{sign}(\langle 2, \mathsf{aenc}(\langle 3, \langle a, n_2\rangle\rangle, b)\rangle, a)\rangle); \\
& \mathsf{rcv}(b,a, & \langle \mathsf{aenc}(\langle 1, \langle a, \langle x_1, x_2\rangle\rangle\rangle, b), \mathsf{sign}(\langle 2, \mathsf{aenc}(\langle 3, \langle a, x_2\rangle\rangle, b)\rangle, a)\rangle); \\
& \mathsf{snd}(b,a, & \langle x_2, \mathsf{senc}(\langle 4, n_3\rangle, x_1)\rangle); \\
& \mathsf{Secret}_1(b,a,b,n_3); & \\
& \mathsf{rcv}(a,b, & \langle n_2, \mathsf{senc}(\langle 4, x_3\rangle, n_1)\rangle)\ ]
\end{array}$$

The execution exec_2 described hereafter is a valid execution of Π_2, *w.r.t.* any intruder knowledge T_0, that violates the secrecy property ϕ_2.

$$
\begin{array}{lll}
\mathsf{exec}_2 = [& \mathsf{snd}(b,a, & \langle \mathsf{aenc}(\langle 5, \langle b, \langle n_4^1, n_3^1\rangle\rangle\rangle, a), \mathsf{sign}(\langle 6, \mathsf{aenc}(\langle 7, \langle b, n_3^1\rangle\rangle, a)\rangle, b)\rangle);\\
& \mathsf{rcv}(a,b, & \langle \mathsf{aenc}(\langle 5, \langle b, \langle n_4^1, n_3^1\rangle\rangle\rangle a), \mathsf{sign}(\langle 6, \mathsf{aenc}(\langle 7, \langle b, n_3^1\rangle\rangle, a)\rangle, b)\rangle);\\
& \mathsf{snd}(a,b, & \mathsf{senc}(\langle 8, n_5^2\rangle, n_4^1));\\
& \mathsf{Secret}_2(a,a,b,n_5^2); & \\
& \mathsf{rcv}(b,a, & \mathsf{senc}(\langle 8, n_5^2\rangle, n_4^1));\\
& \mathsf{snd}(a,b, & \langle \mathsf{aenc}(\langle 1, \langle a, \langle n_1^2, n_2^2\rangle\rangle\rangle, b), \mathsf{sign}(\langle 2, \mathsf{aenc}(\langle 3, \langle a, n_2^2\rangle\rangle, b)\rangle, a)\rangle);\\
& \mathsf{rcv}(b,a, & \langle \mathsf{aenc}(\langle 1, \langle a, \langle n_1^2, n_2^2\rangle\rangle\rangle, b), \mathsf{sign}(\langle 2, \mathsf{aenc}(\langle 3, \langle a, n_2^2\rangle\rangle, b)\rangle, a)\rangle);\\
& \mathsf{snd}(b,a, & \langle \underline{n_2^2}, \mathsf{senc}(\langle 4, n_3^1\rangle, n_1^2)\rangle);\\
& & \\
& \mathsf{snd}(b,a, & \langle \mathsf{aenc}(\langle 5, \langle b, \langle n_4^3, n_3^3\rangle\rangle\rangle, a), \mathsf{sign}(\langle 6, \mathsf{aenc}(\langle 7, \langle b, n_3^3\rangle\rangle, a)\rangle, b)\rangle);\\
& \mathsf{rcv}(a,b, & \langle \mathsf{aenc}(\langle 5, \langle b, \langle n_4^3, n_3^3\rangle\rangle\rangle, a), \mathsf{sign}(\langle 6, \mathsf{aenc}(\langle 7, \langle b, n_3^3\rangle\rangle, a)\rangle, b)\rangle);\\
& \mathsf{snd}(a,b, & \mathsf{senc}(\langle 8, n_5^4\rangle, n_4^3));\\
& \mathsf{rcv}(b,a, & \mathsf{senc}(\langle 8, n_5^4\rangle, n_4^3));\\
& \mathsf{rcv}(b,a, & \langle \mathsf{aenc}(\langle 1, \langle a, \langle \underline{\underline{n_1^\epsilon}}, \underline{n_2^2}\rangle\rangle\rangle, b), \mathsf{sign}(\langle 2, \mathsf{aenc}(\langle 3, \langle a, \underline{n_2^2}\rangle\rangle, b)\rangle, a)\rangle);\\
& \mathsf{snd}(b,a, & \langle n_2^2, \mathsf{senc}(\langle 4, \underline{n_3^3}\rangle, \underline{\underline{n_1^\epsilon}})\rangle);\\
& & \\
& \mathsf{rcv}(a,b, & \langle \mathsf{aenc}(\langle 5, \langle b, \langle \underline{\underline{n_4^\epsilon}}, \underline{n_3^3}\rangle\rangle\rangle, a), \mathsf{sign}(\langle 6, \mathsf{aenc}(\langle 7, \langle b, \underline{n_3^3}\rangle\rangle, a)\rangle, b)\rangle);\\
& \mathsf{snd}(a,b, & \mathsf{senc}(\langle 8, n_5^5\rangle, \underline{\underline{n_4^\epsilon}}));\\
& \mathsf{Secret}_2(a,a,b,n_5^5) &]
\end{array}
$$

By learning the temporary secret n_2^2, the intruder can introduce his nonce n_1^ϵ in key position in the message containing the "temporary secret" n_3^3. In this way, he introduces his key in the message containing the secret n_5^5. This attack requires 5 sessions to be mounted.

This family $\{\Pi_k\}_{k\in\mathbb{N}}$ seems to indicate that the class $\mathbb{P}_{\mathrm{tag}}$ is at the frontier between decidability and undecidability. On the one hand, most existing decidability results, bound the number of sessions that need to be considered (hence the decidability) by a function of the considered formula. But these reduction results do not depend on the "size" of the protocol. The discussion above shows that such a drastic reduction would not be correct for the family $\{\Pi_k\}_{k\in\mathbb{N}}$. On the other hand, all existing undecidability proofs reduce problems well known to be undecidable to the secrecy problem for protocols that do not all admit well-typed attacks. More precisely, such proofs consist of an encoding of the instances of a well know undecidable problem into security protocols. But for the set of resulting protocols the typing abstraction is incorrect in general. So existing undecidability proofs do not apply to the class $\mathbb{P}_{\mathrm{tag}}$.

8. Conclusion

In this paper we presented a reduction result on the type and size of messages to be considered while searching for an attack. We prove this result for the

class of protocols satifying the criterion of "non-unifiable subterms" defined by Abadi and Needham in [AN96]. We showed that a protocol satisfying this simple syntactic criterion admits an attack if and only if it admits a well-typed attack *w.r.t.* a very tight typing system, thus drastically reducing the trace set to be considered. In particular this result implies that only bounded length messages have to be considered. Moreover, we proved that this reduction result holds not only for secrecy but for a large class of properties including several levels of authentication. This result justifies the typing abstraction made by several tools such as [ABB+02, CMR01, BLP06, Cor03]. For those protocols that do not satisfy this condition of non-unifiability, we also prove that a light tagging scheme is sufficient to still be able to apply our results. We furthermore illustrate the fact that the class of protocols considered is quite wide, since the number of sessions required to find an attack is not bounded. In particular we can build a protocol in our class that is flawed but necessitates $2k + 1$ sessions to exhibit an attack.

The work presented above is a step towards the exhibition of a decidable subclass of protocols. Indeed, many proofs of undecidability rely on the fact that an intruder can fool a participant by sending him an encrypted message that has a wrong type without the participant noticing it. One area of interesting future work would be to manage to conclude as to the (un)decidability of the secrecy problem for protocols in $\mathbb{P}$, and in case of undecidability trying to find and study an interesting subclass of $\mathbb{P}$. It would also be interesting to check if our typing abstraction remains sound for trace equivalence properties.

References

[ABB+02] A. Armando, D.A. Basin, M. Bouallagui, Y. Chevalier, L. Compagna, S. Mödersheim, M. Rusinowitch, M. Turuani, L. Viganò, and L. Vigneron. The AVISS security protocol analysis tool. In *Proc. 14th International Conference on Computer Aided Verification(CAV '02)*, volume 2404 of *LNCS*, pages 349–353. Springer, 2002.

[AD07] M. Arapinis and M. Duflot. Bounding messages for free in security protocols. In *Proc. 27th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'07)*, volume 4855 of *LNCS*, pages 376–387. Springer, 2007.

[AN96] M. Abadi and R. M. Needham. Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering*, 22(1):6–15, 1996.

[Bla01] B. Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *Proc. 14th IEEE work. on Computer Security Foundations (CSFW'01)*, page 82. IEEE, 2001.

[BLP06] L. Bozga, Y. Lakhnech, and M. Périn. Pattern-based abstraction for verifying secrecy in protocols. *International Journal on Software Tools for Technology Transfer*, 8(1):57–76, 2006.

[BP05] B. Blanchet and A. Podelski. Verification of Cryptographic Protocols: Tagging Enforces Termination. *Theoretical Computer Science*, 333(1-2):67–90, March 2005. Special issue FoSSaCS'03.

[CC01] H. Comon and V. Cortier. Tree automata with one memory, set constraints and cryptographic protocols. Research Report LSV-01-13, Laboratoire Spécification et Vérification, ENS Cachan, France, 2001. 98 pages.

[CC03] H. Comon-Lundh and V. Cortier. New decidability results for fragments of first-order logic and application to cryptographic protocols. In *Proc. 14th International Conference on Rewriting Techniques and Applications (RTA'03)*, volume 2706 of *LNCS*, pages 148–164. Springer, 2003.

[CDD07] V. Cortier, J. Delaitre, and S. Delaune. Safely composing security protocols. In *Proc. 27th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'07)*, volume 4855 of *LNCS*, pages 352–363. Springer, 2007.

[CDL06] V. Cortier, S. Delaune, and P. Lafourcade. A survey of algebraic properties used in cryptographic protocols. *Journal of Computer Security*, 14(1):1–43, 2006.

[CES06] R. Corin, S. Etalle, and A. Saptawijaya. A logic for constraint-based security protocol analysis. In *Proc. IEEE Symposium on Security and Privacy*, pages 155–168. IEEE, 2006.

[CJ97] J.A. Clark and J.L. Jacob. A survey of authentication protocol literature, 1997. 109 pages. Available at http://www.cs.york.ac.uk/~jac/papers/drareviewps.ps.

[CJM00] E. M. Clarke, S. Jha, and W. Marrero. Verifying security protocols with brutus. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 9(4):443–487, 2000.

[CKR+03] Y. Chevalier, R. Küsters, M. Rusinowitch, M. Turuani, and L. Vigneron. Extending the Dolev-Yao intruder for analyzing an unbounded number of sessions. In *Proc. 17th Int. Work. on Computer Science Logic (CSL03)*, volume 2803 of *LNCS*, pages 128–141. Springer, 2003.

[CMR01] V. Cortier, J. Millen, and H. Rueß. Proving secrecy is easy enough. In *Proc. 14th IEEE work. on Computer Security Foundations (CSFW'01)*, pages 97–106. IEEE, 2001.

[Com04] H. Comon. Résolution de contraintes et recherche d'attaques pour un nombre borné de sessions., 2004. Available at http://www.lsv.ens-cachan.fr/~comon/CRYPTO/bounded.ps.

[Cor03] V. Cortier. A guide for SECURIFY. Technical Report 13, projet RNTL EVA, 2003. 9 pages.

[Cor06] R. Corin. *Analysis Models for Security Protocols*. Phd thesis, University of Twente, 2006.

[CZ06] V. Cortier and E. Zalinescu. Deciding key cycles for security protocols. In *Proc. 13th Inter. Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'06)*, volume 4246 of *LNCS*, pages 317–331. Springer, 2006.

[Del07] S. Delaune. Note: Constraint solving procedure., 2007. Available at http://www.lsv.ens-cachan.fr/Publis/PAPERS/PDF/CDD-fsttcs07-addendum.pdf.

[DLMS99] N. A. Durgin, P. D. Lincoln, J. C. Mitchell, and A. Scedrov. Undecidability of bounded security protocols. In *proceedings of the Workshop on Formal Methods and Security Protocols (FMSP)*, 1999.

[HLS03] J. Heather, G. Lowe, and S. Schneider. How to prevent type flaw attacks on security protocols. *Journal of Computer Security*, 11(2):217–244, 2003.

[HS06] J. Heather and S. Schneider. To infinity and beyond or, avoiding the infinite in security protocol analysis. In *SAC*, pages 346–353, 2006.

[Low97] G. Lowe. A hierarchy of authentication specification. In *Proc. 10th Computer Security Foundations Workshop (CSFW'97)*, pages 31–44. IEEE Computer Society, 1997.

[Low99] G. Lowe. Towards a completeness result for model checking of security protocols. *Journal of Computer Security*, 7(1):89–146, 1999.

[LYH04] Y. Li, W. Yang, and C-W. Huang. Preventing type flaw attacks on security protocols with a simplified tagging scheme. In *Proc. Int. symp. on Information and communication technologies (ISICT'04)*, pages 244–249. ACM, 2004.

[Mea96] C. Meadows. The NRL protocol analyzer: An overview. *Journal of Logic Programming*, 26(2):113–131, 1996.

[MM82] A. Martelli and U. Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(2):258–282, 1982.

[MS01] J. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *Proc. 8th ACM Conference on Computer and Communications Security (CCS'01)*. ACM Press, 2001.

[RS03a] R. Ramanujam and S. P. Suresh. A decidable subclass of unbounded security protocols. In *Proc. Work. on Issues in the Theory of Security (WITS'03)*, 2003.

[RS03b] R. Ramanujam and S. P. Suresh. Tagging makes secrecy decidable with unbounded nonces as well. In *Proc. Foundations of Software Technology and Theoretical Computer Science (FSTTCS'03)*, volume 2914 of *LNCS*, pages 363–374. Springer, 2003.

[RT01] M. Rusinowitch and M. Turuani. Protocol insecurity with finite number of sessions is NP-complete. In *Proc. 14th IEEE work. on Computer Security Foundations (CSFW'01)*, page 174. IEEE, 2001.

[RT03] M. Rusinowitch and M. Turuani. Protocol insecurity with finite number of sessions and composed keys is NP-complete. *Theoretical Computer Science*, 299(1-3):451–475, 2003.

[SBP01] D.X. Song, S. Berezin, and A. Perrig. Athena: a novel approach to efficient automatic security protocol analysis. *Journal of Computer Security*, 9(1-2):47–74, 2001.

Index