Unité Mixte de Recherche 5104 CNRS - INPG - UJF
Centre Equation
2, avenue de VIGNATE
F-38610 GIERES
tel : +33 456 52 03 40
fax : +33 456 52 03 50
http://www-verimag.imag.fr

Pattern-based Abstraction for Verifying Secrecy in Protocols[1]

L. Bozga, Y. Lakhnech and M. Périn

Report n^o 01

October 15, 2004

Pattern-based Abstraction for Verifying Secrecy in Protocols[2]

L. Bozga, Y. Lakhnech and M. Périn

VERIMAG
2, av. de Vignates, 38610 Gières - FRANCE
{fistname.lastname}@imag.fr

October 15, 2004

Abstract

We present a method based on abstract interpretation for verifying secrecy properties of cryptographic protocols. Our method allows to verify secrecy properties in a general model allowing an unbounded number of sessions, an unbounded number of principals and an unbounded size of messages. As abstract domain we use sets of so-called *super terms*. Super terms are obtained by allowing an interpreted constructor, which we denote by *Sup*, where the meaning of a term *Sup* (t) is the set of terms that contain t as sub-term. For these terms, we solve a generalized form of the unification problem and introduce a widening operator. We implemented a prototype and were able to verify well-known protocols such as for instance Needham-Schroeder-Lowe (0.03 sec), Yahalom (12.67 sec), Otway-Rees (0.01 sec) and Kao-Chow (0.78 sec).

Keywords: Security, Cryptographic Protocols, Abstract Interpretation, Verification

Reviewers:

Notes:

How to cite this report:

@techreport { verimag-TR-2004-01,
title = { Pattern-based Abstraction for Verifying Secrecy in Protocols[3]},
authors = { L. Bozga, Y. Lakhnech and M. Périn },
institution = { Verimag Technical Report },
number = {01},
year = { 2004},
note = { }
}

1 Introduction

At the heart of almost every computer security architecture is a set of cryptographic protocols that use cryptography to encrypt and sign data. They are used to exchange confidential data such as pin numbers and passwords, to authentify users or to guarantee anonymity of principals. It is well known that even under the idealized assumption of perfect cryptography, logical flaws in the protocol design may lead to incorrect behavior with undesired consequences. Maybe the most prominent example showing that cryptographic protocols are notoriously difficult to design and test is the Needham-Schroeder protocol for authentication. It has been introduced in 1978 [33]. An attack on this protocol has been found by G. Lowe using the CSP model-checker FDR in 1995 [26]; and this led to a corrected version of the protocol [27]. Consequently there has been a growing interest in developing and applying formal methods for validating cryptographic protocols [29, 15]. Most of this work adopts the so-called Dolev and Yao model of intruders. This model assumes perfect cryptographic primitives and a nondeterministic intruder that has total control of the communication network and has capacity to forge new messages. It is known that reachability is undecidable for cryptographic protocols in the general case [20], even when a bound is put on the size of messages [19]. Because of these negative results, from the point of view of verification, the best we can hope for is either to identify decidable sub-classes as in [5, 35, 30] or to develop correct but incomplete verification algorithms as in [32, 24, 22].

In this paper, we present a correct verification algorithm to prove secrecy without putting any assumption on messages nor on the number of sessions. Proving secrecy means proving that secrets, which are pre-defined messages, are not revealed to unauthorized agents. The main contribution of our paper is a method for proving that a secret is not revealed by a set of rules that model how the protocol extends the set of messages known by the intruder.

Our method is based on the notion of *safe messages that guard a secret* ; these are messages that contain secrets encrypted with safe keys. For example, suppose that our secret is the nonce N_B and that the key K_B^{-1} – the inverse of K_B – is not known by the intruder. We say that K_B is a *safe key*. Then, any message that contains N_B and that is encrypted with K_B is a guard for N_B, e.g., N_B is protected in the message $\{\{N_A, N_B\}_{K_B}\}_{K_I}$ by the safe message $\{N_A, N_B\}_{K_B}$.

Following this idea, given a set K of safe keys we define the K-*guards* as the set of message encrypted with a key in K. However, K-guards can fail at protecting a secret. Indeed, a protocol may reveal some secrets embedded in safe messages. Here is an example from the Needham-Schroeder protocol (see Example 3.1). Consider the action of the responder – played by an honest principal B – in a session between an intruder I and B. The action of B may be seen as a rule $\{I, y\}_{K_B} \to \{y, n_2\}_{K_I}$: On reception of any message matching with the left-hand-side, B will decrypt and send y to the intruder. So, we conclude that the safe key K_B can guard a secret except in messages of the form $\{I, y\}_{K_B}$ where y is a secret.

The idea underlying our verification algorithm is then to characterize the set of K-guards that will keep the secret unrevealed in all sent messages. The K-guards that do not protect their secret are called *safe-breakers*. Let us consider again the Needham-Schroeder protocol and the first transition of principal B described above. Then, $\{I, y\}_{K_B}$ is a safe-breakers.

The core of our verification algorithm takes a protocol and computes an over-approximation of the set of safe-breakers. This set is, in general, infinite. Therefore, we represents it using *terms*: a term with variables represents for the infinite set of its ground instances.

A weakness of this symbolic representation is, however, that variables appear only at the leafs, and hence, they do not allow to describe, for instance, the set of terms that share a common sub-term. To mitigate this weakness, we introduce *super terms*, that is, terms with an interpreted constructor, *Sup*, where a term *Sup* (t) is meant for the set of terms that contain t as sub-term. The use of super terms in our verification method requires to solve a generalized form of the unification problem. In counterpart, it allows us to define a widening operator that ensures termination of a large class of protocols.

We developed a prototype in Caml that implements this method. We have been able to verify several protocols taken from [12] including, for instance, Needham-Schroeder-Lowe (0.03 sec), Yahalom (12.67 sec), Otway-Rees (0.01 sec), and Kao-Chow (0.78 sec).

Related work

Decidability Dolev, Even and Karp introduced the class of ping-pong protocols and showed its decidability. The restriction put on these protocols are, however, too restrictive and none of the protocols of [12] falls in this class. Recently, Comon, Cortier and Mitchell [14] extended this class allowing pairing and binary encryption while the use of nonces still cannot be expressed in their model. Reachability is decidable for the bounded number of sessions [5, 35, 30, 10, 11] or when nonce creation is not allowed and the size of messages is bounded [19]. These assumptions are in practice not always justified.

Security protocols debugging For the general case, model-checking tools have been applied to discover flaws in cryptographic protocols [28, 31, 13]. The tool described in [13] is a model-checker dedicated to cryptographic protocols. Most of these methods bound the number of sessions to be considered as well as the size of the messages.

Deductive methods Methods based on induction and theorem proving have been developed (e.g. [34, 9, 17]). These methods are general, i.e., can handle unbounded protocols, but are not automatic with exception of [17]. This work can be seen as providing a general proof strategy for verifying security protocols. The strategy is implemented on the top of PVS and allows to handle many known protocols. The termination of this strategy is, however, not guaranteed.

Logic programming based methods These methods are based on modeling protocols in Horn Logic, e.g. as Prolog programs, as in [37, 7, 3] and developing suitable proof strategies. The main difficulty in these methods is that termination of the analysis is not guaranteed.

Typing and Abstraction-based methods Type systems and type-checking have also been advocated as a method for verifying security protocols (e.g. [1, 23, 2]). Although, these techniques can handle unbounded protocols they are as far as we know not yet completely automatic. Closest to our work are partial algorithms based on abstract interpretation and tree automata that have been presented in [32, 24, 22, 25]. The main difference is, however, that we do not compute the set of messages that can be known by the intruder but a set of guards as explained above. Our method can handle unbounded protocols fully automatically with the price that it may discover false attacks. Interesting enough is that this does not happen on any of the practical protocols we tried (see Table 8 in Section 7.3). We are actually working on a method that allows to analyze possible attacks.

2 Preliminary

If $n \in \mathbb{N}$ then we denote by $\mathbb{N}_n$ the set $\{1, \cdots, n\}$. Let $\mathcal{X}$ be a countable set of variables and let $\mathcal{F}^i$ be a countable set of function symbols of arity i, for every $i \in \mathbb{N}$. Let $\mathcal{F} = \bigcup_{i \in \mathbb{N}} \mathcal{F}^i$. The set of *terms over $\mathcal{X}$ and $\mathcal{F}$*, denoted by $\mathcal{T}(\mathcal{X}, \mathcal{F})$, is the smallest set containing $\mathcal{X}$ and closed under application of the function symbols in $\mathcal{F}$, i.e., $f(t_1, \cdots, t_n)$ is a term in $\mathcal{T}(\mathcal{X}, \mathcal{F})$, if $t_i \in \mathcal{T}(\mathcal{X}, \mathcal{F})$, for $i = 1, \cdots, n$, and $f \in \mathcal{F}^n$. As usual, function symbols of arity 0 are called constant symbols. *Ground terms* are terms with no variables. We denote by $\mathcal{T}(\mathcal{F})$ the set of ground terms over $\mathcal{F}$.

A tree tr is a function from a non-empty finite subset of ω^* to $\mathcal{X} \cup \mathcal{F}$ such that 1.) if $tr(u) \in \mathcal{F}^n$ then $u \cdot j \in dom(tr)$, for every $j \in \{0, \cdots, n-1\}$ and $u \cdot j \notin dom(tr)$ for every $j \geq n$ and 2.) if $tr(u) \in \mathcal{X}$ then $u \cdot j \notin dom(tr)$ for every $j \in \mathbb{N}$.

We identify terms with trees by associating to each term t a tree $Tr(t)$ as follows:

1. if x is a variable, then $dom(Tr(x)) = \{\varepsilon\}$ and $Tr(x)(\varepsilon) = x$,
2. if $a \in \mathcal{F}^0$ is a constant symbol, then $dom(Tr(a)) = \{\varepsilon\}$ and $Tr(a)(\varepsilon) = a$ and

3. for a term $t = f(t_0, \cdots, t_{n-1})$,$dom(Tr(t)) = \{\varepsilon\} \cup \bigcup_{i=0}^{n-1} i \cdot dom(Tr(t_i))$, where $\cdot$ is word concatenation extended to sets, $Tr(t)(\varepsilon) = f$ and $Tr(t)(i \cdot u) = Tr(t_i)(u)$.

Henceforth, we tacitly identify the term t with $Tr(t)$. The elements of $dom(t)$ are called *positions* in t. We use $\prec$ to denote the prefix relation on ω^*. We write $t(p)$ to denote the symbol at position p in t and $t_{|_p}$ to denote the subterm of t at position p, which corresponds to the tree $t_{|_p}(x) = t(p \cdot x)$ with $x \in dom(t_{|_p})$ iff $p \cdot x \in dom(t)$. We write $q^{-1}p$ to denote the position obtained from p after removing the prefix q. We write $t \preceq t'$ (resp. $t \prec t'$) to denote that t is a sub-term (resp. proper sub-term) of t' . Moreover, $t[t'/p]$ denotes the term obtained from t by substituting t' for $t_{|_p}$. The set of variables in a term t is defined as usual and is denoted by $var(t)$.

3 Models for cryptographic protocols

In this section, we describe how we model cryptographic protocols and give a precise definition of the properties we want to verify. We begin by describing the messages involved in a protocol model.

3.1 Messages

The set of messages is denoted by $\mathcal{T}(\mathcal{F})$ and contains terms constructed from constant symbols and the function symbols **encr** $: \mathcal{T}(\mathcal{F}) \times \mathcal{K} \rightarrow \mathcal{T}(\mathcal{F})$ and **pair** $: \mathcal{T}(\mathcal{F}) \times \mathcal{T}(\mathcal{F}) \rightarrow \mathcal{T}(\mathcal{F})$. Constant symbols are also called atomic messages and are defined as follows:

1. *Principal names* are used to refer to principals in a protocol. The set of all principals is $\mathcal{P}$.
2. *Nonces* can be thought as randomly generated numbers. As no one can predict their values, they are used to convince for the freshness of a message. We denote by $\mathcal{N}$ the set of nonces.
3. *Keys* are used to encrypt messages. An atomic key of the form $f(A_1, \cdots, A_r)$, where f is either pbk, pvk or smk and each A_i is a principal name. Intuitively, pbk, pvk and smk stand respectively for *public*, *private* and *symmetric* keys. The key $pbk(A_1, \cdots, A_r)$ is an inverse of the key $pvk(A_1, \cdots, A_r)$ and vice versa; a key $smk(A_1, \cdots, A_r)$ is its self-inverse. If k is a key then we use k^{-1} to denote its inverse.

   We denote by $\mathcal{AK}(A_1, \cdots, A_r)$ the set of keys described above and let $\mathcal{K} = \bigcup_{\vec{A} \in \mathcal{P}^+} \mathcal{AK}(\vec{A})$ denote the set of all keys.

For the sake of simplicity we left out the signatures and hash functions but we can easily handle them in our model. Let $\mathcal{A} = \mathcal{P} \cup \mathcal{N} \cup \mathcal{K}$ and $\mathcal{F} = \mathcal{A} \cup \{\mathbf{encr}, \mathbf{pair}\}$. As usual, we write (m_1, m_2) for $\mathbf{pair}(m_1, m_2)$ and $\{m\}_k$ instead of $\mathbf{encr}(m, k)$. *Message terms* are the elements of $\mathcal{T}(\mathcal{X}, \mathcal{F})$, that is, terms over the atoms $\mathcal{A}$, a set of variables $\mathcal{X}$ and the binary function symbols **encr** and **pair**. *Messages* are ground terms in $\mathcal{T}(\mathcal{X}, \mathcal{F})$.

Role terms To describe the transitions that can be performed by a principal in a session of a cryptographic protocol, we introduce *role terms*. Let $\mathcal{X}_N$ be a set of variables that range over nonces with $n, n_1, \ldots$ as typical variables and $\mathcal{X}_P$ be a set of variables that range over principals with $p, p_1, \ldots$ as typical variables. We assume that $\mathcal{X}$, $\mathcal{X}_N$ and $\mathcal{X}_P$ are pairwisely disjoint.

Role terms are terms constructed from variables in $\mathcal{X} \cup \mathcal{X}_N \cup \mathcal{X}_P$ using the binary function symbols **encr** and **pair** and where constants are not allowed. More, precisely role terms are defined by the following tree grammar:

$$
\begin{array}{rcl}
\mathit{Key} & ::= & pbk(p_1, \cdots, p_r) \mid pvk(p_1, \cdots, p_r) \\
 & & \mid smk(p_1, \cdots, p_r) \\
\mathit{RT} & ::= & n \mid p \mid \mathit{Key} \mid x \mid \\
 & & \mathbf{pair}(\mathit{RT}_1, \mathit{RT}_2) \mid \mathbf{encr}(\mathit{RT}, \mathit{Key})
\end{array}
$$

where $p, p_1, \cdots, p_r \in \mathcal{X}_P$ and $x \in \mathcal{X}$.

$\mathit{tran}(p_1)$:	$\mathit{tran}(p_2)$:
$- \rightarrow \{(p_1, n_1)\}_{pbk(p_2)}$;	$\{(p_1, y)\}_{pbk(p_2)} \rightarrow \{(y, n_2)\}_{pbk(p_1)}$
$\{(n_1, z)\}_{pbk(p_1)} \rightarrow \{z\}_{pbk(p_2)}$	

Figure 1: Needham-Schreoder protocol

3.2 Cryptographic Protocols - Syntax

To describe cryptographic protocols, we need to describe the transitions the principals can perform. In our setting, transitions have the form $t \rightarrow t'$, where t and t' are role terms with $\mathit{var}\,(t') \subseteq \mathit{var}\,(t)$, t is called the *guard* of the transition and t' its *action*.

Now, a cryptographic protocol is described by a parameterized session description where the parameters are the involved principals, the fresh nonces and used keys. A *session description* is then given by a tuple $(P, \mathit{tran}, \mathit{fresh})$, where

- P is a vector $(p_1, \cdots, p_r)$, $r \geq 1$, of distinct principal variables in $\mathcal{X}_P$,
- *tran* is a function that associates to each principal variable in P a finite sequence of transitions,
- *fresh* associates to each principal variable p in P a disjoint finite set of nonce variables in $\mathcal{X}_N$. By abuse of notation we sometimes write $\mathit{fresh}(P)$ to denote $\bigcup_{p \in P} fresh(p)$.

Example 3.1 The Needham-Schroeder protocol for authentication can be described as follows using the usual informal notation for cryptographic protocols:

$$\begin{array}{lcl} A \rightarrow B & : & \{A, N_1\}_{k_B} \\ B \rightarrow A & : & \{N_1, N_2\}_{k_A} \\ A \rightarrow B & : & \{N_2\}_{k_B} \end{array}$$

Intuitively, A plays the role of the initiator of the session; while B is a responder. In our setting it is described by the session description given in Figure 1, where $P = (p_1, p_2)$, $\mathit{fresh}(p_1) = \{n_1\}$ and $\mathit{fresh}(p_2) = \{n_2\}$. As one can see, our description is much more detailed and elevates many of the ambiguities of the informal description. □

3.3 The intruder model

In this section, we describe how an intruder can create new messages from already known messages. We use the most commonly used model, introduced by Dolev and Yao [18], which is given by a formal system $\vdash$. The intruder capabilities for intercepting messages and sending (fake) messages are fixed by the operational semantics. Thus, the *derivability of a message* m from a set E of messages, denoted by $E \vdash m$, is described by the following axiom and rules:

- If $m \in E$ then $E \vdash m$.
- If $E \vdash m_1$ and $E \vdash m_2$ then $E \vdash \mathbf{pair}(m_1, m_2)$. This rule is called pairing.
- If $E \vdash m$ and $E \vdash k \in \mathcal{K}$ then $E \vdash \mathbf{encr}(m, k)$. This is called encryption.
- If $E \vdash \mathbf{pair}(m_1, m_2)$ then $E \vdash m_1$ and $E \vdash m_2$. This is called projection.
- If $E \vdash \mathbf{encr}(m, k)$, $E \vdash k'$ and k and k' are inverses then $E \vdash m$. This is called decryption.

Pairing and encryption rules are called *composition* rules while projection and decryption are called *decomposition* rules. As usual, derivations in the system $\vdash$ can be seen as proof trees.

For a set of messages M, we use the notation $E \vdash M$ to denote $E \vdash m$ for each m in M and $E \not\vdash M$ to denote $E \not\vdash m$ for each m in M.

It is worth noticing that the intruder cannot forge any key term from the knowledge of its subterms, e.g, $A, B \not\vdash smk(A, B)$. No rules are provided to the intruder to do so. Consequently, from the intruder point of view, the key terms are atomic keys.

Critical and non-critical positions Since there is no way to deduce the key used for encryption from an encrypted message we consider their positions not critical, *ie.* it is a safe place for a secret. For instance, the position of the key k for the **encr** constructor – as in the term **encr**(m, k) – is not critical ; on the other hand the position of m is critical. The critical position corresponds to the subterm relation in the strand space model [36, 21].

Formally, given a term t, a position p in t is called *non-critical*, if there is a position q such that $t(q) =$ **encr** and $p = q \cdot 1$; otherwise it is called *critical*. We will also use the notation $s \in_c m$ to denote that s appears in m at a critical position, i.e., there exists $p \in dom(m)$ such that p is critical and $m_{|_p} = s$.

For a term t, we use the notation $E \not\vdash t$ to denote that no instance of t is derivable from E, that is, for no substitution $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F})$, we have $E \vdash \sigma(t)$.

We also use the notation $E \not\vdash^{\in_c} t$ to denote that no message derivable from E contains an instance of t at a critical position, that is, for every message m and ground substitution σ, if $E \vdash m$ then $\sigma(t) \notin_c m$. The relation $\not\vdash^{\in_c}$ is naturally extended to sets of terms.

3.3.1 Operational semantics

In the rest of this section, let $\mathcal{S} = (P, tran, fresh)$ be a given session description. We want to describe the behavior of the protocol described by $\mathcal{S}$ without any restriction on the numbers of sessions and principals. To do so, we need to define *instantiated transitions* and *instantiated sessions*. We use natural numbers as session identifiers.

Session instances A session instance is fixed by a pair (i, π), where i is its identifier and π is a vector of principals that instantiate the principal variables $p_1, \cdots, p_r$. Therefore, we introduce the set $\text{Inst} = \mathbb{N} \times \mathcal{P}^r$ of session instances. As we impose that the principal variables in P are distinct, we can use $\pi(p_j)$ to refer the j^{th} principal name in the vector π, i.e., we can identify π with a function, $\pi : P \to \mathcal{P}$. We refer to a session instance by its identifier.

We assume that we have for each fresh variable $n \in fresh(P)$ an injective function which associates for each session instance a fresh nonce value, $n : \mathbb{N} \to \mathcal{N}$ such that $n_1(i_1) \neq n_2(i_2)$, if $n_1 \neq n_2$ or $i_1 \neq i_2$. That is, any fresh parameter is instantiated with different values in different sessions. Moreover, different fresh parameters are instantiated with different values in the same or in different sessions. We write N^i for the value of $n(i)$ where n is a nonce fresh variable and i is the session instance identifier. Intuitively, we use N^i as the nonce corresponding to the fresh variable n in the session instance (i, π).

In order to produce an instance of the session description we have to choose a fresh session number and a substitution that associates a constant name to each principal variable in P. Hence, given $(i, \pi) \in \text{Inst}$, we generate a session instance, denoted by $(\mathcal{S})^i_\pi$, by applying the following transformations to all role terms that appear in $\mathcal{S}$:

- we replace each principal variable p by $\pi(p)$,
- each nonce variable $n \in fresh(P)$ by $N^{i,\pi}$.

We denote by t^i_π the message term obtained from t by applying the transformations above. Then, the (i, π)-*instance* of a transition $t \to t'$ is $t^i_\pi \to t'^i_\pi$. Given $p \in P$, we denote by $tran^i_\pi(p)$ the sequence of (i, π)-instantiated transitions obtained from $tran(p)$.

Example 3.2 Let $(i = 0, \pi = (A, B))$ be a session instance. Moreover, $n_1(0) = N_1$ and $n_2(0) = N_2$. Then, the $(0, \pi)$-instance of the Needham-Schroeder protocol contains the transitions given in Figure 2.

$tran^0_\pi(A)$:	$tran^0_\pi(B)$:
$- \quad \rightarrow \{(A, N_1)\}_{pbk(B)}$;	$\{(A, y)\}_{pbk(B)} \rightarrow \{(y, N_2)\}_{pbk(A)}$
$\{(N_1, z)\}_{pbk(A)} \rightarrow \{z\}_{pbk(B)}$	

Figure 2: The transitions of the $(0, \pi)$-instance.

Configurations and transitions In order to define global configurations that may arise during the protocol execution, we need to define the state of each session instance.

The *state* of a session instance S_i is given by a pair (π, τ), where τ associates for each role of the protocol $p \in P$ the sequence of its instantiated actions, which are left to be executed. Initially in the session instance S_i, $\tau(p) = tran^i_\pi(p)$.

The configurations set of the protocol defined by $\mathcal{S}$ is given by a pair (ξ, E), where $dom(\xi) = \mathbb{N}$ is the set of identifiers of the sessions created in the configuration, $\xi(i)$ describes the state of the session instance S_i and E is a set of messages. The operational semantics is defined as a labelled transition system over the set of configurations. There are two sets of transitions:

1. *transitions that create new sessions:*

$$\frac{i \notin dom(\xi)}{(\xi, E) \rightarrow (\xi[i \mapsto (\pi, \tau)], E)}$$

   where τ is a function that associates for each role of the protocol $p \in P$ the sequence of instantiated actions $\tau(p) = tran^i_\pi(p)$ and π is an arbitrary assignment of principals to parameters. That corresponds to creating a new session instance (i, π).

2. *transitions that correspond to transition inside sessions:*

$$\frac{(\tau, E) \Rightarrow (\tau', E')}{(\xi, E) \rightarrow (\xi[i \mapsto (\pi, \tau')], E')}$$

   where $\xi(i) = (\pi, \tau)$ and $\Rightarrow$ is defined below.

   The relation $\Rightarrow$ describes session state changes caused by firing principal transitions. We have $(\tau, E) \Rightarrow (\tau', E')$, if there is $t \rightarrow t'$ which is the first transition in $\tau(p)$ for $p \in P$ and there is a substitution $\rho : \mathcal{X} \rightarrow \mathcal{T}(\mathcal{F})$ such that $E \vdash t\rho$. $E' = E \cup \{t'\rho\}$ and $\tau' = \tau[p \mapsto tail(\tau(p))]$. Where $tail$ function returns all but the first element in a sequence.

Example 3.3 Consider again our running example, the Needham-Schroeder protocol, and a session between *A* and *B*, identified by 0, with principal *A* in the last step of the protocol. Hence, $\pi(p_1) = A$ and $\pi(p_2) = B$ and the state $\xi(0)$ of the session is (π, τ) where $\tau(p_1) = \{(N_1, z)\}_{K_A} \rightarrow \{z\}_{K_B}$ and $\tau(p_2) = \epsilon$. Here, we use the shorter notation K_A for $pbk(A)$.

Moreover, let $\{N_1, N_2\}_{K_A} \in E$ then *A* can fire its last transition which modifies the session state: $(\tau, E) \rightarrow (\tau', E')$, where $\tau'(p_1) = \tau'(p_2) = \epsilon$ and $E' = E \cup \{\{N_2\}_{K_B}\}$.

Clarifying remarks In our model the intruder has the ability to intercept any message sent by a principal and principals have no guarantee about the origin of a message. Thus, the intruder can intercept messages, use them to create fake messages and deliver these to the principals. Following Bolignano [8], in our model this is realized by modeling sending of messages as adding messages to the set E and by modeling receiving of messages as reading messages deducible from E. Principals use, however, the guards of the transitions to check the genuineness of received messages. For instance, in the Needham-Schroeder example, the guard $\{(p_1, y)\}_{pbk(p_2)}$ of the transition of principal p_2 means that principal p_2 accepts any and only messages that are pair with p_1 in the first position and encrypted by $pbk(p_2)$. Consider now the guard of the second transition of p_1, namely $\{(n_1, z)\}_{pbk(p_1)}$. Here, p_1 refuses (and the execution blocks) if the message to be read is not an encryption by $pbk(p_1)$ of a pair whose first message is the nonce n_1 sent in the first transition.

3.4 Secrecy modeling

A *secrecy* goal states that a designated message should not be made public. A secret is public when it is deducible from the set of messages intercepted by the intruder. In our setting, a secret is defined by a role term. For instance, in the Needham-Schroeder example a secret we want to prove is n_2, the nonce sent by p_2. More precisely, each session instance is associated with a secret we want to prove. Here arises the important question concerning the initial knowledge of the intruder and his ability to profit from the actions of honest participants in parallel and previous sessions. In other words, when proving that the secret associated to session i running between the participants A and B remains unrevealed, we have to take into account that an intruder can profit from a session between A and C to break the protocol. Actually, we cannot even rely on the honesty of C; she can be seen as an intruder's accomplice.

As in the previous section, let $\mathcal{S} = (P, \mathit{tran}, \mathit{fresh})$ be a given session description.

A *secret template* is given by a role term s. Given $(i, \pi) \in \text{Inst}$ we denote by $C(E, \pi, i)$ the constraint stating that the intruder cannot initially know messages that contain fresh nonces, private keys or symetric keys of the principals in π.

Moreover, let $C(E)$ denote the condition:

$$\forall (i, \pi) \in \text{Inst} \cdot C(E, \pi, i).$$

We are now ready to define our secrecy property formally. A protocol described by $\mathcal{S}$ satisfies the secrecy property defined by the secret template s in the initial set E_0 of intruder's messages, denoted by *Secret*$(\mathcal{S}, s, E_0)$ or $\not\vdash_P s$, if for every $(i, \pi) \in \text{Inst}$ if $C(E_0)$, $(\emptyset, E_0) \rightarrow^* (\xi, E)$ and $\xi(i) = (\pi, \tau)$ then $E \not\vdash s^i_\pi$. The definition of secrecy can be easily extended to a set S of secret templates by:
Secret$(\mathcal{S}, S, E_0)$ iff *Secret*$(\mathcal{S}, s, E_0)$, for all $s \in S$.

4 Finite abstraction of atomic messages and sessions

In this section we fix an arbitrary cryptographic protocol given by a session description $\mathcal{S} = (P, \mathit{tran}, \mathit{fresh})$ and fix a secret s given by a role term. To prove that s is a secret, we are faced with the following problems:

1. The definition of our verification problem is a reachability problem quantified universally over all $(i, \pi) \in$ Inst.

2. There is no bound on the number of sessions that can be created.

3. There is no bound on the size of the messages that occur during execution of the protocol.

In this section, we present an abstraction that copes with the first two problems. The other problem is handled in the next section. We proceed in two steps. First, we present an abstraction that is parameterized by $(i_0, \pi_0) \in \text{Inst}$, then we argue that the abstract system we obtain does not depend on the choice of (i_0, π_0). The main idea of the abstraction is as follows. Clearly, the behavior of a participant does not depend on its identity. This is simply a consequence of defining protocol sessions in a parameterized manner as we did. It also does not depend on the identifier associated to the session.

Therefore, we fix an arbitrary session where the participants, say we have two, are A and B. Then, we identify with the intruder I all participants other than A and B. Moreover, we identify all sessions in which neither A nor B are involved. Concerning the other sessions, that is, those where A or B are involved, we identify:

- all sessions where A plays the role of p_1, B plays the role of p_2 and the session is different from the fixed session,

- all sessions where B plays the role of p_1 and A plays the role of p_2,

- all sessions where A plays the role of p_1 and the role of p_2 is played by a participant different from A or B,

- all sessions where B plays the role of p_1 and the role of p_2 is played by a participant different from A or B, etc
- all sessions where A plays the role of p_2 and the role of p_1 is played by a participant different from A or B,
- all sessions where B plays the role of p_2 and the role of p_1 is played by a participant different from A or B.

Identifying sessions means also identifying the nonces and keys used in these sessions. This leaves us with a system where we have a finite number of participants, of nonces and of keys but an unbounded number of sessions. Therefore, we apply an abstraction that removes the control. To summarize, we model a protocol as a set of transitions that can be taken in any order and any number of times. The number of messages as their size are left not bounded.

Furthermore, we consider only two principals, one honest principal A and one dishonest principal, the intruder I. That this abstraction is safe and complete is proved in [16].

We now present this idea formally. Let $(i_0, \pi_0) \in$ Inst be fixed. For a concrete semantic object x, we use the notation $x^{(i_0,\pi_0)}$ to denote its abstraction, and in case (i_0, π_0) is known from the context we use $x^\sharp$.

We start by defining the abstract domains $\mathbb{N}^\sharp = \{\top, \bot\}$ and $\mathcal{P}^\sharp = \{A, I\}$ and the abstractions:

- $i^\sharp = \begin{cases} \top & \text{if } (i,\pi) = (i_0,\pi_0) \\ \bot & \text{otherwise} \end{cases}$
- $p^\sharp = \begin{cases} A & \text{if } p = \pi_0(p_i), p_i \in P \\ I & \text{otherwise} \end{cases}$

We extend the abstraction of participants to vectors of participants by taking the abstractions of the components.

The abstraction of the nonce N^i, of a session instance (i, π), denoted by $(N^i)^\sharp$, is given by:

- N_I, if $n \in fresh(p)$ and $\pi(p)^\sharp = I$,
- N, if $i^\sharp = \top$, and
- $N^{\pi^\sharp}$, otherwise.

where N_I is a fresh constant.

Thus, as abstract sets of nonce, we have $\mathcal{N}^\sharp(I) = \{N_I\}$ and $\mathcal{N}^\sharp(A_j) = \{N, N^{\pi^\sharp} \mid n \in fresh(p_j), \pi(p_j) = A_j\}$.

Example 4.1 For Needham-Schroeder, we have the following set of abstract nonces:

$$\mathcal{N}^\sharp = \{N_I, N_1, N_2, {N_1}^{A,x}, {N_2}^{x,A} \mid x \in \{A, I\}\}.$$

We denote $\mathcal{N}^\sharp = \mathcal{N}^\sharp(I) \cup \bigcup\limits_{j \in \mathbb{N}_r} \mathcal{N}^\sharp(A_j)$.

It remains to define the abstraction of keys. We take the abstract set $\mathcal{K}^\sharp$ that consists of a distinguished key K_I and the keys in $\mathcal{AK}(p_1^\sharp, \cdots, p_l^\sharp)$ with $p_1^\sharp, \cdots, p_l^\sharp \in \mathcal{P}^\sharp$ and $p_j^\sharp \neq I$, for all $j \in \mathbb{N}_l$. The abstraction of a key $k(p_1, \cdots, p_n)$ is defined by:

$$k^\sharp(p_1, \ .., p_n) = \begin{cases} k(p_1^\sharp, \cdots, p_n^\sharp) & \\ \quad \text{if } p_i^\sharp \neq I,\ i = 1, \cdots, n & \\ K_I \quad \text{otherwise} \end{cases}$$

Example 4.2 For Needham-Schroeder, we have the following set of abstract keys:

$$\mathcal{K}^\sharp = \{K_I, pbk(A), pvk(A)\}.$$

We denote $\mathcal{A}^\sharp = \mathcal{P}^\sharp \cup \mathcal{N}^\sharp \cup \mathcal{K}^\sharp$.

The abstraction of a message term t, denoted by $t^\sharp$, is obtained as the homomorphic extension of the abstractions on participants, nonces and keys. For a set T of terms, let $T^\sharp = \{t^\sharp \mid t \in T\}$.

The set $\mathcal{T}(\mathcal{F})^\sharp$ of abstract messages is the set of ground terms over $\mathcal{A}^\sharp$ and the constructors **encr** and **pair** as for $\mathcal{T}(\mathcal{F})$. Similarly, we can define the set of abstract terms by allowing variables in $\mathcal{X}$.

We are now ready to define the abstraction of a cryptographic protocol that will be given as a pair $(C^\sharp, R)$, where $C^\sharp$ is a set of constraints of the form $E^\sharp \not\vdash^{\in_c} m^\sharp$, with $m \in \mathcal{T}(\mathcal{F})^\sharp$ and $E \subseteq \mathcal{T}(\mathcal{F})^\sharp$ and R is a set of abstract transitions. We call $(C^\sharp, R)$ an *abstract protocol*. The pair $(C^\sharp, R)$ defines a transition system whose initial states are sets $E^\sharp \subseteq \mathcal{T}(\mathcal{F})^\sharp$ that satisfy $C^\sharp$ and where we have $E^\sharp \rightarrow_R E^{\sharp\prime}$, if there is $t \rightarrow t'$ in R and $\rho : \mathcal{X} \rightarrow \mathcal{T}(\mathcal{F})^\sharp$ such that $E^\sharp \vdash \rho(t)$ and $E^{\sharp\prime} = E^\sharp \cup \{\rho(t')\}$.

The abstraction $\mathcal{S}^\sharp$ of the cryptographic protocol defined by $\mathcal{S}$ is defined by:

- $C^\sharp$ is the abstraction of $C(E, \pi_0, i_0)$ and
- the set R of abstract transitions $t_1^\sharp \rightarrow_R t_2^\sharp$ such that $t_1 \rightarrow t_2$ is a transition in some session instance $\mathcal{S}_\pi^i$.

We also call R abstract transitions rules. Let $\mathcal{ST} = (C^\sharp, R)$ be an abstract protocol and $E_0^\sharp \subseteq \mathcal{T}(\mathcal{F})^\sharp$. We say that $\mathcal{ST}$ *preserves the secret* $s^\sharp$ *in* $E_0^\sharp$, denoted by $E_0^\sharp \not\vdash_{\mathcal{ST}} s^\sharp$, if for all $E^\sharp \subseteq \mathcal{T}(\mathcal{F})^\sharp$, if $C^\sharp(E_0^\sharp)$ and $E_0^\sharp \rightarrow_R^* E^\sharp$ then $E^\sharp \not\vdash s^\sharp$.

To relate a cryptographic protocol and its abstraction, we need to relate derivation by the intruder on the concrete and abstract messages. We can prove by structural induction on m the following:

Lemma 4.1 *Let E be a set of messages and $E^\sharp = \{m^\sharp \mid m \in E\}$. Then, $E \vdash m$ implies $E^\sharp \vdash m^\sharp$, for any message $m \in \mathcal{T}(\mathcal{F})$.* □

Proof We prove by induction on the tree derivation

1. $E \vdash m$ in one step: Hence, $m \in E$. By the definition of $E^\sharp = \{m^\sharp \mid m \in E\}$ then $m^\sharp \in E^\sharp$ and then $E^\sharp \vdash m^\sharp$.
2. Induction step. $E \vdash m$ in $k+1$ steps. We make a case analysis on the last derivation step:
   - Case of pairing, $m = (m_1, m_2)$. We have $E \vdash m_1$ and $E \vdash m_2$ in k steps, then by induction hypothesis $E^\sharp \vdash m_1^\sharp$ and $E^\sharp \vdash m_2^\sharp$ and by **pair** rule we have $E^\sharp \vdash (m_1^\sharp, m_2^\sharp)$ but $(m_1^\sharp, m_2^\sharp) = (m_1, m_2)^\sharp$ so $E^\sharp \vdash (m_1, m_2)^\sharp$.
   - Case of encryption, $m = \{t\}_k$. Similarly to the previous case.
   - Case of left projection. We have $E \vdash (m, m')$ in k steps, then by induction hypothesis $E^\sharp \vdash (m, m')^\sharp$ but $(m, m')^\sharp = (m^\sharp, m'^\sharp)$ so $E^\sharp \vdash (m^\sharp, m'^\sharp)$ and by left projection rule we have $E^\sharp \vdash m^\sharp$. Similarly for right projection.
   - Case of decryption. We have $E \vdash \{m\}_k$ and $E \vdash inv(k)$ in k steps, then by induction hypothesis $E^\sharp \vdash \{m\}_k^\sharp$ and $E^\sharp \vdash (k^{-1})^\sharp$ which is equivalent with $E^\sharp \vdash \{m^\sharp\}_{k^\sharp}$ and $E^\sharp \vdash (k^\sharp)^{-1}$ then by decryption rule we have $E^\sharp \vdash m^\sharp$.

□

We can also prove the following lemma to relate concrete and abstract term instantiations:

Lemma 4.2 *Let t_1 and t_2 be two terms and let $\rho : \mathcal{X} \rightarrow \mathcal{T}(\mathcal{F})$ be a ground substitution. Then, $\rho(t_1) = \rho(t_2)$ implies $\rho^\sharp(t_1^\sharp) = \rho^\sharp(t_2^\sharp)$, where $\rho^\sharp(X)$ is defined as $\rho(X)^\sharp$.* □

Proof

We have $\rho(t_1) = \rho(t_2)$ implies $\rho(t_1)^\sharp = \rho(t_2)^\sharp$ and we prove by structural induction on the term t that $\rho(t)^\sharp = \rho^\sharp(t^\sharp)$:

1. case t atomic - by definition

2. case $t = f(t_1, t_2)$, where $f \in \{\mathbf{pair}, \mathbf{encr}\}$:

$$\begin{array}{lll}
\rho(t)^\sharp & = & \rho(f(t_1,t_2))^\sharp \\
& = & f(\rho(t_1),\rho(t_2))^\sharp \\
& = & f(\rho(t_1)^\sharp,\rho(t_2)^\sharp) \\
& = & f(\rho^\sharp(t_1^\sharp),\rho^\sharp(t_2^\sharp)) \text{ by induction hypothesis} \\
& = & \rho^\sharp(f(t_1^\sharp,t_2^\sharp)) \\
& = & \rho^\sharp(f(t_1,t_2)^\sharp) \\
& = & \rho^\sharp(t^\sharp)
\end{array}$$

□

Using Lemma 4.1 and Lemma 4.2, we can prove that $(C^\sharp, R)$ is indeed an abstraction of $\mathcal{S}$ where the abstraction of a configuration (ξ, E) is $E^\sharp$:

Proposition 4.1 *Let $\mathcal{S} = (P, \text{tran}, \text{fresh})$ be a protocol and $\mathcal{S}^\sharp = (C^\sharp, R)$ its abstraction. Let (ξ_1, E_1) and (ξ_2, E_2) be concrete configurations Then,*

$$(\xi_1, E_1) \to (\xi_2, E_2) \textit{ implies } E_1^\sharp \to_R E_2^\sharp.$$

Moreover, if $C(E)$ is true then also $C^\sharp(E^\sharp)$. □

Proof Following the protocol transitions we have two cases:

1. transition that create new session (i, π):
   We have $E_1 = E_2$ and then $E_1^\sharp \to_R E_2^\sharp$.
2. inside session transition $(t \to t') \in$ *tran*:
   We have $(\xi_1, E_1) \to (\xi_2, E_2)$ where $E_2 = E_1 \cup (t'\sigma)$ and σ is a substitution $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F})$ such that $E \vdash t\sigma$. We will prove that there is an abstract transition in R such that $E_1^\sharp \to_R E_1^\sharp \cup \{(t'\sigma)^\sharp\}$
   By lemma 4.1 we have $E \vdash t\sigma \Rightarrow E^\sharp \vdash (t\sigma)^\sharp \quad (*)$
   Also, since $(t \to t') \in$ *tran* in the abstract transition R of the protocol we have $t^\sharp \to t'^\sharp$. Then, from $(*)$ using the lemma 4.2 we obtain
   $E_1^\sharp \to_R (E_1^\sharp \cup \{(t'\sigma)^\sharp\}$, hence $E_1^\sharp \to_R E_2^\sharp$

□

Exploiting Proposition 4.1 and the fact that $(C^\sharp, R)$ does not depend on (i_0, π_0), that is, we have the same constraints and transitions for all $(i, \pi) \in$ Inst, we can prove:

Corollary 4.1 *The protocol defined by $\mathcal{S}$ satisfies the secrecy property defined by S in E_0, if its abstraction $(C^\sharp, R)$ preserves $S^\sharp$ in $E_0^\sharp$, i.e.,*

$$E_0^\sharp \not\vdash_{(C^\sharp,R)} S^\sharp \textit{ implies Secret}(\mathcal{S}, S, E_0).$$

Example 4.3 In our model which yields an over-approximation of the possible runs of the protocol, we can describe the Needham-Schroeder protocol by the rules of Figure 3.

In this form, the relation between the message expected to fire a transition and the corresponding answer is made explicit through variables. Each rule of a session corresponds to a transition of the Needham-Schroeder protocol as shown in figure 3 in which the roles and nonces are instantiated w.r.t. the principals of the session. Additionally, a verification tool requires a constraint $C(E)$ on the initial knowledge of the intruder defined by $E \not\vdash^{\in_c} \{\ N_1,\ N_2,\ pvk(A)\ \}$ and a secrecy property defined by the set of messages $\{N_2, pvk(A)\}$.

Sessions	Transitions		
a fixed sess. (A, A)	$- \rightarrow \{A, N_1\}_{pbk(A)}$	$\{A, y\}_{pbk(A)} \rightarrow \{y, N_2\}_{pbk(A)}$	$\{N_1, z\}_{pbk(A)} \rightarrow \{z\}_{pbk(A)}$
other sessions (A, A)	$- \rightarrow \{A, N_1^{AA}\}_{pbk(A)}$	$\{A, y\}_{pbk(A)} \rightarrow \{y, N_2^{AA}\}_{pbk(A)}$	$\{N_1^{AA}, z\}_{pbk(A)} \rightarrow \{z\}_{pbk(A)}$
the sessions (A, I)	$- \rightarrow \{A, N_1^{AI}\}_{pbk(I)}$	$\{A, y\}_{pbk(I)} \rightarrow \{y, N_I\}_{pbk(A)}$	$\{N_1^{AI}, z\}_{pbk(A)} \rightarrow \{z\}_{pbk(I)}$
the sessions (I, A)	$- \rightarrow \{I, N_I\}_{pbk(A)}$	$\{I, y\}_{pbk(A)} \rightarrow \{y, N_2^{IA}\}_{pbk(I)}$	$\{N_I, z\}_{pbk(I)} \rightarrow \{z\}_{pbk(A)}$

Figure 3: The abstract rules of Needham-Schroeder Protocol

5 The verification method

Throughout this section we assume that we are given a protocol $P = (\mathcal{C}, \mathcal{R})$ and a set of secrets defined by a set $\mathcal{S}$ of messages. We present an algorithm that allows to verify that a protocol preserves a set of secrets. If a principal A wants to protect a secret s, he has to encrypt every occurrence of s in every message sent with a key whose inverse is not known by the intruder. The secret s itself need not to be directly encrypted; it is enough that the secret only appears as part of encrypted messages.

The basic idea of our method is to compute the set of encrypted messages that protect the secrets. As we will see, encryption with a safe key is not always sufficient to protect a secret in every message. The honest principals following the protocol can unwittingly help the intruder in decrypting messages.

In order to develop this idea formally we need to introduce a few definitions. In the sequel, we let $K \subseteq \mathcal{K}$ denote a fixed but arbitrary set of keys and we assume $\emptyset \neq K \neq \mathcal{K}$. Keys in K are *safe keys*, i.e., keys whose inverses are not known by the intruder and therefore protect m. We call K**-guard** any encrypted message $\{m\}_k \in \mathcal{T}(\mathcal{F})$ where k is a safe key. We call **safe-breaker** a pair $(\{m\}_k, p)$, where $\{m\}_k$ is a K-guard and p is a critical position[4] in $\{m\}_k$. Intuitively, p denotes the position of a secret and a safe-breaker $(\{m\}_k, p)$ means that, in the specific case of message $\{m\}_k$, the intruder can pass through the protection of key k and obtain the sub-term at position p.

Definition 5.1 *Let m and s be two messages, let $\mathcal{B}$ be a set of safe-breakers and let K be a set of safe keys. We denote by $\neg\big(m\langle\mathcal{B}\rangle_K s\big)$ (or $\neg m\langle\mathcal{B}\rangle_K s$ for readability) the predicate "s is reachable in m by application of some safe-breakers in $\mathcal{B}$ together with the intruder's decomposition rules". We use the predicate postively and negatively. The positive version of the predicate, $m\langle\mathcal{B}\rangle_K s$, can be red as "the secret s is insensitive to $\mathcal{B}$ in a message m". For the sake of simplicity, we define the negation of the predicate $m\langle\mathcal{B}\rangle_K s$ by the following inference rules:*

$$\frac{}{\neg m\langle\mathcal{B}\rangle_K m} \qquad \frac{\neg m\langle\mathcal{B}\rangle_K s,\ k \notin K}{\neg\{m\}_k\langle\mathcal{B}\rangle_K s}$$

$$\frac{\neg m_1\langle\mathcal{B}\rangle_K s}{\neg(m_1, m_2)\langle\mathcal{B}\rangle_K s} \qquad \frac{\neg m_2\langle\mathcal{B}\rangle_K s}{\neg(m_1, m_2)\langle\mathcal{B}\rangle_K s}$$

$$\frac{k \in K,\ \neg(\{m\}_k)_{|_p}\langle\mathcal{B}\rangle_K s,\ (\{m\}_k, p) \in \mathcal{B}}{\neg\{m\}_k\langle\mathcal{B}\rangle_K s}$$

This definition is easily generalized to sets of messages: Let M and $\mathcal{S}$ be sets of messages, and $\mathcal{B}$ a set of safe-breakers. We say that the secrets $\mathcal{S}$ are insensitive to $\mathcal{B}$ in M, denoted by $M\langle\mathcal{B}\rangle_K\mathcal{S}$, if $\forall m \in M, \forall s \in \mathcal{S}.\ m\langle\mathcal{B}\rangle_K s$. Morevoer, a secret of $\mathcal{S}$ is reachable in M with the help of safe-breakers $\mathcal{B}$, denoted by $\neg M\langle\mathcal{B}\rangle_K\mathcal{S}$, if $\exists m \in M, \exists s \in \mathcal{S}.\ \neg m\langle\mathcal{B}\rangle_K s$.

Example 5.1 Let $m = \mathbf{pair}(A,\ \{A, \{N\}_{k_1}\}_{k_2})$, and let k_1 and k_2 be two safe keys, i.e. $\{k_1, k_2\} \subseteq K$. Then, $m\langle\emptyset\rangle_K N$ holds, meaning that N is not deducible from m without safe-breaker. Indeed, the intruder would not gain anything in splitting the pair, since N is protected in both parts: $A\langle\emptyset\rangle_K N$ and $\{A, \{N\}_{k_1}\}_{k_2}\langle\emptyset\rangle_K N$ hold.

[4]Critical and non-critical positions as well as the notation $\in_c$ are introduced in Section 3.3.

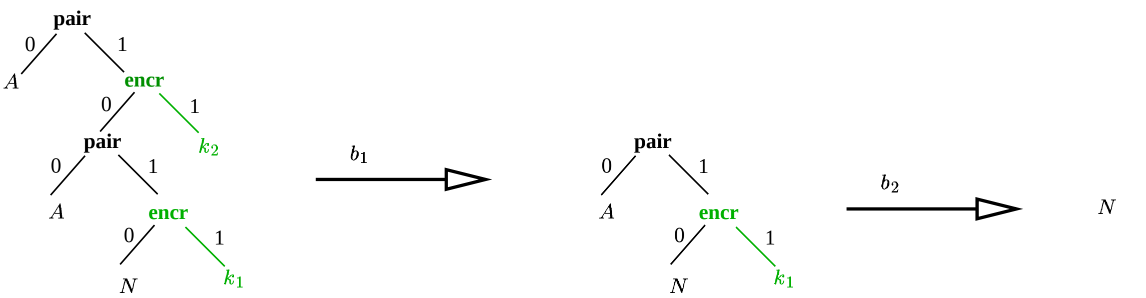

Figure 4: Example 7 : Application of safe-breakers

Let $b_1 = (\{A, \{N\}_{k_1}\}_{k_2}, 01$ and $b_2 = (\{N\}_{k_1}, 0)$ be two safe-breakers. Let $\mathcal{B} = \{\, b_1, b_2 \,\}$. Then, $m\langle\mathcal{B}\rangle_K N$ does not hold, meaning that the safe-breakers can be applied to get the secret N. This is illustrated by Figure 5.1. Indeed, by Definition 5.1, $m\langle\mathcal{B}\rangle_K N$ is true if and only if we have $A\langle\mathcal{B}\rangle_K N$ and $\{A, \{N\}_{k_1}\}_{k_2}\langle\mathcal{B}\rangle_K N$. The former one holds, but this is not the case of the latter one: an application of the first safe-breaker provides $\{N\}_{k_1} = \{A, \{N\}_{k_1}\}_{k_2}{}_{|01}$. Then, an application of the second safe-breaker provides $N = (\{N\}_{k_1})_{|0}$. Since $N\langle\mathcal{B}\rangle_K N$ does not hold (this is the case where $m = s$), Definition 5.1 entails $\neg\big(m\langle\mathcal{B}\rangle_K N\big)$.

The notion of a message insensitive to safe-breakers does not take into account the capabilities of the intruder to decompose and compose new messages.

Example 5.2 Consider the set of messages $E = \{s_1, s_2\}$. Whatever $\mathcal{B}$ we choose, the property $E\langle\mathcal{B}\rangle_K (s_1, s_2)$ trivially holds since (s_1, s_2) does not belong to E. However, the pair (s_1, s_2) can be derived from E using the pairing rule.

This example shows that we have to give particular care to the treatment of composed secrets as they can be obtained either by composition or decomposition. To do so, we define the closure under decomposition of a term. Taking the closure of a set $\mathcal{S}$ of secrets ensures that the intruder cannot derive a message in $\mathcal{S}$ solely by composition rules.

Let M be a set of sets of messages and let m be a message. We say that M is *closed* w.r.t. m, if it consists of all messages on some path of m. We denote by $\mathrm{c}(m)$ the set of all sets of messages closed w.r.t. m.

Then, a set M of messages is *closed against composition*, if for any $m \in M$ there exists a set of messages $M' \in \mathrm{c}(m)$ such that $M' \subseteq M$.

Example 5.3 Consider the message $m = (\{(A, N)\}_k, B)$. The sets closed w.r.t. are the following:

$$
\begin{array}{l}
\{(\{(A,N)\}_k, B), B\};\\
\{(\{(A,N)\}_k, B), \{(A,N)\}_k, k\};\\
\{(\{(A,N)\}_k, B), \{(A,N)\}_k, (A,N), A\};\\
\{(\{(A,N)\}_k, B), \{(A,N)\}_k, (A,N), N\}
\end{array}
$$

The closure computation helps in preventing the intruder from making m by composition: it tells us that it is sufficient to ensure that one of these sets of messages remains completely unknown to the intruder.

We can prove the following:

Lemma 5.1 *Let $\mathcal{S}$ and E be two sets of messages such that $\mathcal{S} \cap E = \emptyset$, and assume $\mathcal{S}$ is closed against composition. Then, no message in $\mathcal{S}$ can be derived using only composition rules. In symbols we write $E \nvdash_c \mathcal{S}$ where $\vdash_c$ denotes a derivation that use only composition rules.*

Our purpose now is to define conditions such that for any set E of messages, if the secrets of $\mathcal{S}$ are insensitive to safe-breakers in the set of messages E, then the secrets are protected in all messages derivable from E. In other words, we look for a condition that ensures the stability of protection under the derivation rules that define Dolev and Yao's intruder.

Example 5.4 Consider the set of messages $E = \left\{k_2, \{s\}_{k_1}\right\}$. The safe-breaker $(\,\{\{s\}_{k_1}\}_{k_2}, 00\,)$ does not help getting the secret s since it can not be applied to any message of E E does not contain the message $\{\{s\}_{k_1}\}_{k_2}$. Therefore, $E\langle(\,\{\{s\}_{k_1}\}_{k_2}, 00\,)\rangle_K s$ holds.
However, the term $\{\{s\}_{k_1}\}_{k_2}$ is derivable from E using the encryption rule and then, the safe-breaker can be used to get the secret s. □

In order to catch this ability of the intruder – to forge a message and to bring principals to play some transitions that decompose the message – we define a closure on safe-breakers that enriches the set of safe-breakers with their sub-encrypted-messages.

Let (b, p) be a safe-breaker and let *ssb*(b, p) denote the *sub-safe-breakers* of (b, p), that is the set of all *proper sub-terms of* b that are safe-breakers for position p. A formal definition of the function *ssb* is given in Appendix A but let us give an intuitive example.

Example 5.5 Consider two keys $k_1, k_2 \in K$, the message $b = \{\,(\{N\}_{k_1}, A)\,\}_{k_2}$, and assume N at position 000 in b is the secret. Then, by definitions, b is a K-guard and the pair $(b, 000)$ denotes a safe-breaker for N in b. Moreover, each encryption with a key in K in b that is above $N = b_{|000}$ defines a K-guard of N. The function *ssb* computes the position of N in each of these K-guard and returns the set of safe-breakers associated to theses K-guards. For instance, *ssb*$(b, 000)$ returns $\left\{\,(\{N\}_{k_1}, 0)\,\right\}$; and both *ssb*$(b, 01)$ and *ssb*$(b, 00)$ return $\emptyset$, since there is no K-guard that is a proper sub-term of b and above $b_{|01} = A$ (resp. $b_{|00} = \{N\}_{k_1}$).

We are now able to express the conditions that guarantee stability of the predicate $E\langle\mathcal{B}\rangle_K \mathcal{S}$ under the deduction rules of the intruder. In the rest of the paper, $\mathcal{B}$ denotes a set safe-breakers and $\mathcal{S}$ denotes a set of secrets.

Definition 5.2 *A pair* $(\mathcal{B}, \mathcal{S})$ *is* well-formed *with respect to a set of safe keys* K*, if the following conditions are satisfied:*

1. $\mathcal{S}$ *is closed against composition,*
2. $K^{-1} = \{k^{-1} \mid k \in K\} \subseteq \mathcal{S}$*, that is, the inverse of the safe keys are secrets,*
3. *For any safe-breakers* $(b, p) \in \mathcal{B}$*, all its sub-safe-breakers already belong to* $\mathcal{B}$*. Formally,* $\forall(b, p) \in \mathcal{B}.\, \forall(b', p') \in ssb(b, p) \cdot (b', p') \in \mathcal{B}$.

Intuitively, Condition (1) ensures that the intruder will always miss at least one part of a composed secret preventing him from deducing it by composition. Condition (2) ensures that the intruder will not be able to decrypt a secret protected by a key of K. The last condition of well-formedness takes into account the ability of the intruder to use encryption in order to obtain a message that can be broken using a safe-breaker.

The main property of the predicate $E\langle\mathcal{B}\rangle_K \mathcal{S}$ is that it is stable under the intruder's deduction rules.

Proposition 5.1 *Let* E *be a set of messages and* $(\mathcal{B}, \mathcal{S})$ *be a pair of safe-breakers and secrets. If* $(\mathcal{B}, \mathcal{S})$ *is* well-formed *and* $E\langle\mathcal{B}\rangle_K \mathcal{S}$ *holds, then the secrets of* $\mathcal{S}$ *are insensitive to* $\mathcal{B}$ *in any message* m *derivable from* E*, that is,* $E \vdash m \Rightarrow m\langle\mathcal{B}\rangle_K \mathcal{S}$.

Proof See Appendix B.1. □

The following corollary is an immediate consequence of Proposition 5.1.

Corollary 5.1 *If* $E\langle\mathcal{B}\rangle_K \mathcal{S}$ *and* $(\mathcal{B}, \mathcal{S})$ *is well-formed then* $E \nvdash \mathcal{S}$.

Under well-formedness of $(\mathcal{B}, \mathcal{S})$, the predicate $E\langle\mathcal{B}\rangle_K \mathcal{S}$ is stable w.r.t. to the intruder inference system. We now come to the computation of a well-formed pair $(\mathcal{B}, \mathcal{S})$ that ensures in addition the stability of $E\langle\mathcal{B}\rangle_K \mathcal{S}$ w.r.t. any interleaving of sessions of a given protocol $P = (\mathcal{C}, \mathcal{R})$.

Definition 5.3 (stability of $(\mathcal{B}, \mathcal{S})$ w.r.t. to rules) *Let* $r = t_1 \to t_2$ *be a rule in* $\mathcal{R}$*. The pair* $(\mathcal{B}, \mathcal{S})$ *is* stable w.r.t. the rule r*, if for every substitution* σ*, the property* $\sigma(t_1)\langle\mathcal{B}\rangle_K \mathcal{S}$ *implies* $\sigma(t_2)\langle\mathcal{B}\rangle_K \mathcal{S}$*. A pair* $(\mathcal{B}, \mathcal{S})$ *is* stable w.r.t. a set of rules $\mathcal{R}$ *if it is stable w.r.t. to each rule in* $\mathcal{R}$.

The stability of the pair $(\mathcal{B}, \mathcal{S})$ w.r.t. to a rule $t_1 \to t_2$ expresses the fact that the message produced by firing the transition $t_1 \to t_2$ has no effect on the protection of $\mathcal{S}$. Then, using Proposition 5.1, we can prove by induction the following theorem:

Theorem 5.1 *Let $\mathcal{S}$ be a set of secrets and $\mathcal{B}$ be a set of safe-breakers. If $(\mathcal{B}, \mathcal{S})$ is well-formed and stable w.r.t. all rules in $\mathcal{R}$; if additionally, $E_0 \langle \mathcal{B} \rangle_K \mathcal{S}$ holds for every set of messages E_0 that satisfies $\mathcal{C}$, then $\not\vdash_P \mathcal{S}$, i.e., the secrets in $\mathcal{S}$ are preserved in any execution of the protocol $P = (\mathcal{C}, \mathcal{R})$.*

Proof See Appendix B.2. □

Theorem 5.1 gives a sufficient condition to conclude that the secrets in $\mathcal{S}$ are preserved in spite of the protocol $P = (\mathcal{C}, \mathcal{R})$. Given a protocol $P = (\mathcal{C}, \mathcal{R})$ and a set $\mathcal{S}$ of secrets, we compute a set $\mathcal{B}$ of safe-breakers and a set $\mathcal{S}'$ of secrets such that:

- the set of messages initially known by the intruder – defined by the constraint $\mathcal{C}$ on E_0 – satisfies $E_0 \langle \mathcal{B} \rangle_K \mathcal{S}'$,
- $\mathcal{S} \subseteq \mathcal{S}'$, and
- $(\mathcal{B}, \mathcal{S}')$ is well-formed,
- $(\mathcal{B}, \mathcal{S}')$ is stable w.r.t. $\mathcal{R}$.

6 Computing stable secrets and safe-breakers

In this section, we develop an algorithm that computes a stable pair $(\mathcal{B}, \mathcal{S}')$. This is done in two steps. First, we develop a semantic version of the algorithm in which we do not consider questions related to representing sets of safe-breakers. Then, we define a symbolic representation for safe-breakers and we develop a symbolic algorithm.

6.1 A semantic verification algorithm

In Figure 5, we present an algorithm that computes a pair $(\mathcal{B}, \mathcal{S})$ which is well-formed, and stable w.r.t. the rules of the protocol. The algorithm uses a function *Closure* that when applied to a set of messages yields a closure of this set. That is, we describe an algorithm that is parameterized by a choice of such a function. Its correction does not depend in this choice. In fact, we can integrate computing the closure of sets into the algorithm and we can for a given set try all possible closure sets. This is, however, cumbersome and does not add new insight. The algorithm takes as input: a set of rules $\mathcal{R}$, a set of secrets $\mathcal{S}$, a set of safe keys K and a set of safe-breakers $\mathcal{B}$. It is a fixpoint computation of a well-formed stable pair, starting with $(\mathcal{B}, \mathcal{S})$. If it terminates, it returns an augmented set of secrets $\mathcal{S}'$ and an augmented set of safe-breakers $\mathcal{B}'$.

We now explain intuitively the clue point of the algorithm. Let us take a rule $t_p \to t_c$ in $\mathcal{R}$, a substitution $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F})$ such that a secret s is insensitive to $\mathcal{B}$ in $\sigma(t_p)$, the premise of the instantiated rule. If the secret s is not protected in $\sigma(t_c)$, the conclusion of the instantiated rule, then each K-guard of $\sigma(t_p)$ that protect an occurrence of the secret s is not efficient in this case and it must be added to the set of safe-breakers. Indeed, the intruder does not need the inverse of the keys in K to get the secret: it will be unwittingly revealed by a principal who plays the rule $\sigma(t_p) \to \sigma(t_c)$. Think for instance of a protocol with $\{(y, x)\}_{pbk(A)} \to \{x\}_{pbk(y)}$ as a rule of principal A. The principal A will respond $\{Secret\}_{pbk(I)}$ on reception of the message $\{(I, Secret)\}_{pbk(A)}$. Thus unwittingly decrypting the secret for the intruder. So, the K-guard $\{(I, Secret)\}_{pbk(A)}$ is a particular case where $pbk(A)$ does not protect the secret and $\left(\{(I, Secret)\}_{pbk(A)}, 01\right)$ must be added to the set the safe-breakers $\mathcal{B}$. Case 2 in the algorithm considers the case where a secret is vulnerable to safe-breakers in the conclusion, and the premise does not contain a secret. In this case, the apparently harmless premise is as compromising as the secret, and so, it must be added to the set of secrets. The following proposition summarizes the properties of the algorithm.

Proposition 6.1 *If the algorithm of Figure 5 applied to $(\mathcal{R}, \mathcal{S}, K, \mathcal{B})$ terminates, it returns $\mathcal{S}'$ and $\mathcal{B}'$ that satisfy the following conditions:*

input: $\mathcal{R}$, $\mathcal{S}$, K and $\mathcal{B}$
output: $\mathcal{B}'$, $\mathcal{S}'$ such that $(\mathcal{B}', \mathcal{S}')$ is well-formed and stable w.r.t. $\mathcal{R}$.
$\mathcal{B}' := \mathcal{B}$; $\mathcal{S}' := \mathcal{S}$;
— add to the secrets the inverse of the keys from K
$K^{-1} = \{k^{-1} \mid k \in K\}; \mathcal{S}' := \mathcal{S}' \cup K^{-1}$;
repeat
— compute the closure that adds to S' one subpart of each compound secret of $\mathcal{S}'$
$\mathcal{S}' := Closure(\mathcal{S}')$; $\mathcal{B}_c := \mathcal{B}'$; $\mathcal{S}_c := \mathcal{S}'$;
for each $t_p \to t_c \in \mathcal{R}$
for each $r \in dom(t_c)\ s.\ t.\ (t_c)_{|_r} \in \mathcal{X} \cup \mathcal{S}$
— compute all Dangerous Substitutions of rule $t_p \to t_c$ where a secret is
— not kept in the conclusion
$DS := \{\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F}) \mid \neg\big(\sigma(t_c)\langle\mathcal{B}'\rangle_K (t_c)_{|_r}\big) \wedge \exists s \in \mathcal{S} s.t.\neg\big(\sigma((t_c)_{|_r})\langle\mathcal{B}'\rangle_K s\big)\}$;
— compute the corresponding Dangerous Premises
$DP := \{\sigma(t_p) \mid \sigma \in DS\}$;
— update the secret and safe-breakers according to the dangerous premises:
— case 1 add safe-breakers to $\mathcal{B}'$ if $(t_c)_{|_r} \in_c t_p$
for each $m \in DP$ **do**
— new safe-breakers are pairs constructed from submessage of m of the form $\mathbf{encr}(m', k),\ k \in K$
— and positions of $(t_c)_{|_r}$ in them

$$new\mathcal{B} := \{(m_{|_q}, q^{-1}r') \mid \exists k \in K,\ m_{|_q} = \{m_{|_{q\cdot 0}}\}_k \wedge r'\ critical\ position\ s.\ t.\ (t_p)_{|_{r'}} = (t_c)_{|_r} \wedge q \prec r'\}$$

— update the set of safe-breakers $\mathcal{B}$
$\mathcal{B}' := \mathcal{B}' \cup new\mathcal{B}$;
od
— case 2 adds to the secrets all dangerous premises if $(t_c)_{|_r} \notin_c t_p$
$newS := \{m \mid m \in DP\}$; $\mathcal{S}' := \mathcal{S}' \cup newS$
od
od
until $(\mathcal{B}', \mathcal{S}') = (\mathcal{B}_c, \mathcal{S}_c)$

Figure 5: The semantic version of the verification algorithm

1. $(\mathcal{B}', \mathcal{S}')$ *is well-formed,*
2. $(\mathcal{B}', \mathcal{S}')$ *is stable w.r.t.* $\mathcal{R}$, *and*
3. $\mathcal{S} \subseteq \mathcal{S}'$

Proof The well-formed property of $(\mathcal{B}', \mathcal{S}')$ derives directly from the operations made in the algorithm. First, the set of secrets $\mathcal{S}'$ is each time closed. Second, any time a dangerous premise with respect to a secret s is found we add to $\mathcal{B}'$, all message transducers obtained by its subterms of the form $\{m\}_k$, with $k \in K$ that dominates the secret s and the related positions. Hence, that ensures the second condition of the well-formedness.

If the algorithm reaches a fixpoint $(\mathcal{B}', \mathcal{S}')$ then, the **until** condition of **repeat** termination will be reached. That is, all rule in R produce dangerous substitution DS which generates $new\mathcal{B}$ and $newS$ which are already in $\mathcal{B}'$ respectively $\mathcal{S}'$.

Since we start with the set $\mathcal{S}' = \mathcal{S}$, and then the algorithm only augments it, the last condition, $\mathcal{S} \subseteq \mathcal{S}'$ is obviously satisfied. □

Using Proposition 6.1 and Theorem 5.1, we can prove the following corollary.

Corollary 6.1 *If the algorithm of Figure 5 terminates with* $(\mathcal{B}', \mathcal{S}')$ *as result, and each set of messages* E_0 *that satisfies* $\mathcal{C}(E_0)$ *also satisfies* $E_0\langle\mathcal{B}'\rangle_K \mathcal{S}'$, *we can conclude* $\not\vdash_P \mathcal{S}'$, *and hence,* $\not\vdash_P \mathcal{S}$.

6.2 A symbolic representation of safe-breakers

To develop an effective version of our semantic algorithm, we need to represent (potentially infinite) sets of safe-breakers. To do so, we introduce a symbolic representation of safe-breakers: *a breaking-pattern* is a pair $(\{t\}_k, p)$ where $\{t\}_k$ is a term over variables in $\mathcal{X}$ and p is a critical position in $\{t\}_k$. A secret s embedded in a message m is insensitive to a breaking-pattern (b, p) if it is insensitive to any instance of the pattern b, meaning that the following property holds:

$$m\langle\ \{(\sigma(b), p) \mid \sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F})\}\ \rangle_K\, s$$

For instance, the messages $\{(Secret, (B, A))\}_K$ and $\{(A, (B, Secret))\}_K$ are insensitive to the breaking-pattern[5] $(\{(A, (x, y))\}_K, 010)$, while the secret of message $\{(A, (Secret, B))\}_K$ is revealed by applying the breaking-pattern with the substitution $[x \leftarrow Secret, y \leftarrow B]$.

A breaking-pattern is then a symbolic representation of a set of safe-breakers. In fact, the symbolic algorithm deals with sets of breaking-patterns. So, we go one step further and we introduce *super terms* to represent sets of breaking-patterns. Let us now define formally those symbolic representations used in the HERMES tool, our implementation of the symbolic algorithm.

The *super terms* are defined by the following BNF:

$$\begin{aligned} st \quad ::= \quad & N \mid P \mid K \mid x \mid \mathbf{pair}(st_1, st_2) \mid \\ & \mathbf{encr}(st, K) \mid Sup\,(st) \end{aligned}$$

where $N \in \mathcal{N}$, $P \in \mathcal{P}$, $K \in \mathcal{K}$, and $x \in \mathcal{X}$. The set of super terms is denoted by $\mathcal{ST}(\mathcal{X}, \mathcal{F})$. Notice that every term in $\mathcal{T}(\mathcal{X}, \mathcal{F})$ is also a super term in $\mathcal{ST}(\mathcal{X}, \mathcal{F})$. The difference between the two is that super terms make use of the special Sup function symbol.

Intuitively, as can be seen from the following definition, $Sup\,(t)$ represents all terms containing the term t as a sub-term. For instance, the terms A, $\mathbf{pair}(x, A)$, $\mathbf{encr}(A, K)$, $\cdots$ all belong to $[\![Sup\,(A)]\!]$.

Definition 6.1 *Given a super term* st, *the set of all corresponding terms is denoted by* $[\![st]\!]$. *It is defined as follows:*

$$\begin{aligned} &[\![st]\!] = \{st\} \ \text{ if } st \text{ is a constant or a variable} \\ &[\![\mathbf{pair}(st_1, st_2)]\!] = \{\mathbf{pair}(t_1, t_2) \mid t_1 \in [\![st_1]\!], t_2 \in [\![st_2]\!]\} \\ &[\![\mathbf{encr}(st_1, k)]\!] = \{\mathbf{encr}(t_1, k) \mid t_1 \in [\![st_1]\!]\} \\ &[\![Sup\,(st)]\!] = \{\ t \in \mathcal{T}(\mathcal{X}, \mathcal{F}) \mid \exists \text{ a position } p \text{ in } t \\ &\qquad\qquad\qquad\qquad \text{such that } t_{|p} \in [\![st]\!]\ \} \end{aligned}$$

[5] 010 is the position of x in $\{(A, (x, y))\}_K$

Definition 6.2 *Given a super term st, and a critical position p, we denote by $[\![(st, p)]\!]$ the set of* breaking-term *associated to the* breaking-super term (st, p). *We overload the function $[\![\cdot]\!]$ for the meaning is clear from its argument. For breaking-super terms, the function $[\![\cdot]\!]$ is defined as follows:*

$$
\begin{array}{l}
[\![(st, p)]\!] = \{(st, p)\} \quad \textit{if } st \textit{ is a constant or a variable} \\
[\![(\mathbf{pair}(st_1, st_2), \epsilon)]\!] = \\
\quad \{(\mathbf{pair}(t_1, t_2), \epsilon) \mid t_1 \in [\![st_1]\!], t_2 \in [\![st_2]\!]\} \\
[\![(\mathbf{pair}(st_1, st_2), 0.p)]\!] = \\
\quad \{(\mathbf{pair}(t_1, t_2), 0.q) \mid (t_1, q) \in [\![(st_1, p)]\!],\ t_2 \in [\![st_2]\!]\} \\
[\![(\mathbf{pair}(st_1, st_2), 1.p)]\!] = \\
\quad \{(\mathbf{pair}(t_1, t_2), 1.q) \mid t_1 \in [\![st_1]\!],\ (t_2, q) \in [\![(st_2, p)]\!]\} \\
[\![(\mathbf{encr}(st, k), \epsilon)]\!] = \{(\mathbf{encr}(t, k), \epsilon) \mid t \in [\![st]\!]\} \\
[\![(\mathbf{encr}(st, k), 0.p)]\!] = \{(\mathbf{encr}(t, k), 0.q) \mid (t, q) \in [\![(st, p)]\!]\} \\
[\![(\mathbf{encr}(st, k), 1.p)]\!] = \emptyset \\
[\![(\mathit{Sup}\,(st), \epsilon)]\!] = \emptyset \\
[\![(\mathit{Sup}\,(st), 0.p)]\!] = \{(t, q.r) \mid (t_{|_q}, r) \in [\![(st, p)]\!]\} \\
[\![(\mathit{Sup}\,(st), 1.p)]\!] = \emptyset
\end{array}
$$

Example 6.1 The super term $\big(\mathit{Sup}\,(\mathbf{pair}(A, x)), 01\big)$ denotes all breaking-pattern (b, p) that contain $\mathbf{pair}(A, x)$ as a sub-message of b and where p corresponds to the position of x. The computation of the breaking-patterns corresponding to the super term $\big(\mathit{Sup}\,(\mathbf{pair}(A, x)), 01\big)$ goes through the step $[\![\big(\mathbf{pair}(A, x), 1\big)]\!] = \{\ \big(\mathbf{pair}(A, x), 1\big)\ \}$ and ends with the set $[\![\big(\mathit{Sup}\,(\mathbf{pair}(A, x)), 01\big)]\!] = \{\ (t, q.1) \mid t_{|_q} = \mathbf{pair}(A, x)\ \}$. This set contains for instance the terms $\big(\mathbf{pair}(\mathbf{pair}(A, x), B), 01\big)$, $\big(\mathbf{pair}(B, \mathbf{pair}(A, x)), 11\big)$, $\big(\mathbf{pair}(A, x), 1\big)$, $\big(\mathbf{encr}(\mathbf{pair}(B, \mathbf{pair}(A, x)), k), 111\big)$. □

Using the function $[\![\cdot]\!]$ we can shift from super terms to their equivalent representation of sets of terms. Based on that remark, we present the algorithm on terms and we explain how it extends to super terms. In the sequel, when there is no need to distinguish between terms and super terms, we use the generic word "pattern".

Based on the symbolic representation, the infinite set $\mathcal{B}$ of safe-breakers is represented by a finite set of breaking-patterns $\mathcal{BP}$. More formally, we have the following:

Definition 6.3 *A* symbolic representation SR *is a pair $(\mathcal{BP}, \mathcal{S})$, where*

- *$\mathcal{BP}$ is a finite set of breaking-patterns that represents the safe-breakers $\mathcal{B}$*
- *$\mathcal{S}$ is a finite set of terms that represents the secrets.*

7 A symbolic verification algorithm

The symbolic algorithm is obtained from the algorithm of Figure 5 by replacing each operation by a corresponding symbolic one that operates on $(\mathcal{BP}, \mathcal{S})$. For the sake of presentation, first we explain the symbolic algorithm in the particular case where the breaking-patterns consists of pairs of terms and positions rather than super terms and positions, i.e., *Sup* does not occur in any breaking-patterns of $\mathcal{BP}$. We will explain later how it extends to super terms and what are the difficulties to solve.

7.1 The algorithm on terms

Before presenting the algorithm we need to introduce the following definitions. As usual a substitution is a mapping $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{X}, \mathcal{F})$. A ground substitution is a mapping $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F})$. Let $bp = (t, p)$ and $bp' = (t', p')$ be two breaking-patterns. We say that they unify if the positions p and p' are comparable and there is a substitution $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{X}, \mathcal{F})$ such that $\sigma(t) = \sigma(t')$. We write, also, $\sigma(t, p) = \sigma(t', p')$.

The symbolic algorithm takes as input a set of rules $\mathcal{R}$, a set of secrets $\mathcal{S}$, a set of key K and an empty set of breaking-patterns $\mathcal{BP} = \emptyset$. It computes a new well-formed pair of breaking-patterns and secrets $(\mathcal{BP}, \mathcal{S})$ until it becomes stable w.r.t. all rules in $\mathcal{R}$. Let us now sketch its main steps:

1. The set $\mathcal{S}$ of secrets is augmented with K^{-1}, the set of keys of the form k^{-1} such that k is an element of K.

2. For each rule $t_p \to t_c$ in $\mathcal{R}$, we have to consider all possible occurrences of a secret in the conclusion t_c. So, for each position p in t_c that corresponds to a variable or a secret the algorithm computes:

   a. the finite set of dangerous substitution DS is as follows. A substitution $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{X}, \mathcal{F})$ is *dangerous* if for every position $q \prec p$, for which $\exists k \in K$ such that $(t_c)_{|_q} = \{(t_c)_{|_{q \cdot 0}}\}_k$, the safe-breaker $((t_c)_{|_q}, q^{-1}p)$ unifies by σ with a breaking-pattern of $\mathcal{BP}$. Then, $DS := \{\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{X}, \mathcal{F}) \mid \sigma \textit{ is dangerous}\}$. We illustrate below the computation of dangerous substitutions.
   The set of *dangerous premises* is: $DP = \{\sigma(t_p) \mid \sigma \in DS\}$.

   b. If there exists q such that $(t_c)_{|_p} = (t_p)_{|_q}$ then for every term t of DP we construct a set of new breaking-patterns that consist in pairs of sub-terms of t that are encrypted term by keys from K and positions restricted to this sub-terms of q.
   Formally,
   $$new\mathcal{BP} = \{(t_{|_r}, r^{-1}q) \mid t \in DP, \exists k \in K, t_{|_r} = \{t_{|_{r \cdot 0}}\}_k \wedge\ (t_c)_{|_p} = (t_p)_{|_q} \wedge r \prec q\}.$$
   Update the breaking-patterns
   $\mathcal{BP} := \mathcal{BP} \cup new\mathcal{BP}$.

   c. Otherwise, if such a q does not exists then the set of dangerous premises must be added to the set of secrets. Formally, $new\mathcal{S} := \{m \mid m \in DP\}$. Update and at the same time closure the set of secrets $\mathcal{S} = Closure(\mathcal{S} \cup new\mathcal{S})$.

3. repeat 2 until $new\mathcal{S} \subseteq \mathcal{S}$ and $new\mathcal{BP} \subseteq \mathcal{BP}$.

Computation of dangerous substitutions

We present the algorithm that computes the dangerous substitutions induced by a rule $t_p \to t_c$, and a position p. Let K be the fixed set of keys and $\mathcal{BP}$ the set of breaking-patterns.

Let $\mathcal{PP}$ be the set of positions p_i above p such that for each $p_i \in \mathcal{PP}$ there is $k \in K$ such that $(t_c)_{|_{p_i}} = \{(t_c)_{|_{p_i \cdot 0}}\}_k$. We define below the function Φ that computes all the unifiers between breaking-patterns of $\mathcal{BP}$ and (t_c, p) that cancel each protecting position. Formally, the dangerous substitutions are the unifiers σ that satisfy:
$\bigwedge_{p_i \in \mathcal{PP}} \sigma((t_c)_{|_{p_i}}, p_i^{-1}p) = \sigma(b_i, q_i)$, where
$(bi, pi) \in \mathcal{BP}$.

Initially, Φ is called with the set $\mathcal{PP}$ of protecting positions and a set of substitutions DS containing only the empty substitution: $DS = \{[\,]\}$. Then, it takes in turn each protecting position and if it is possible, it completes the substitutions of DS in order to cancel the current position by a breaking-pattern of $\mathcal{BP}$.

$$\Phi(t_c,\ \mathcal{BP},\ \mathcal{PP},\ DS) = \begin{cases} \{\sigma \mid \sigma \in DS\} & \text{if } \mathcal{PP} = \emptyset \\ \Phi(t_c,\ \mathcal{BP},\ \mathcal{PP} \setminus \{p_i\}, & \\ \quad \bigcup_{\sigma_j \in DS} \{\sigma_j \cup \sigma_1^{i,j}, \cdots, \sigma_j \cup \sigma_{n_{i,j}}^{i,j}\}\ ), & p_i \in \mathcal{PP} \end{cases}$$

where the $\sigma_k^{i,j}$ in the fourth argument are the unifiers resulting of the unification of $(\sigma_j(t_c)_{|_{p_i}}, p_i^{-1}p)$ with some breaking-patterns of $\mathcal{BP}$.

The same algorithm is used in the case where breaking-patterns are pairs of terms and positions and when breaking-patterns are pairs of super terms and positions. We only need to adapt the unification algorithm. In the case of terms, we use the standard *most general unifier*; and for super terms, we define a unification algorithm presented in Section 7.2.

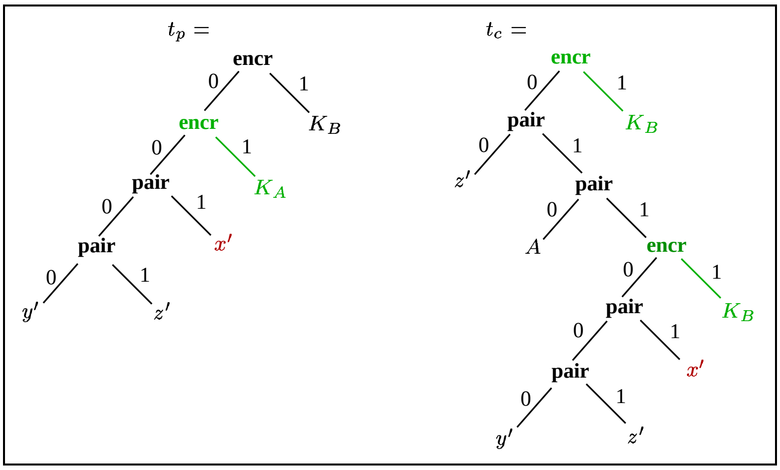

Figure 6: Illustration of computing dangerous substitutions.

Example 7.1 We illustrate the computation of dangerous substitutions on the set of breaking-patterns $\mathcal{BP} = \{\ (\{(I,x)\}_{K_B}, 01),\ (\{((A,y),z)\}_{K_B}, 01)\ \}$, the set of key $K = \{K_A, K_B\}$ and a rule $t_p \to t_c$ given in Figure 6.

We consider the conclusion of the rule. The first step consists in looking for all the critical positions in the conclusion where a secret or a variable appears. We find x' at position 01101, y' at position 011000 and z' at positions 00, 011001 in the term t_c. Let take the position $p = 01101$ of x', we look for the positions above it that may protect it. We found exactly two protecting positions: $p_1 = \epsilon$ and $p_2 = 011$. Then, the function Φ looks for all substitutions that unify some breaking-patterns of $\mathcal{BP}$ with the terms at the protecting positions p_1 and p_2 and the restricted respective positions of p. Starting with position $p_1 = \epsilon$, it unifies $((t_c)_{|\epsilon}, p)$ with the breaking-pattern
$(\{(I,x)\}_{K_B}, 01)$ we have $01 \prec p$ and the unifier $\sigma' = [z' = I, x = (t_c)_{|01}]$. This cancels the top most protection. Then, the function Φ attempts to complete the substitution σ' so that it also cancels the protection at position $p_2 = 011$. To do so, it tries to unify $(\sigma'(t_c)_{|p_2} = \{((y', I), x')\}_{K_B}, p_2^{-1}p = 01)$ with some breaking-patterns of $\mathcal{BP}$ and succeeds with the breaking-pattern $(\{((A,y),z)\}_{K_B}, 01)$. We have $01 = 01$ and the unifier $\sigma'' = [y' = A, y = I, x' = z]$. The two unifiers are then composed and restricted to the domain *var* (t_c) resulting the substitution $\sigma = (\sigma' \cup \sigma'')_{/var(t_c)} = [y' = A, z' = I]$. Pursuing the computation does not provide other substitutions and finally Φ returns for the position p of t_c the set of dangerous substitutions $\{\sigma\}$. We now look at the premise of the rule to compute the new breaking-patterns induced by σ. The variable x' appears in t_p at the position $q = 001$ and it is protected by the key K_B at the position $r_1 = \epsilon$ and by the key K_A at the position $r_2 = 0$. However, the dangerous substitution σ tells that these protections will not work in case y is A and z is I. Consequently, we increase the set of breaking-patterns $\mathcal{BP}$ by adding these particular cases. In our symbolic representation, this means to add

1. $\sigma(t_p) = \{\{((A,I),x')\}_{K_A}\}_{K_B}), r1^{-1}q = 0001$ and
2. $\sigma(t_p)_{|0} = \{((A,I),x')\}_{K_A}, r_2^{-1}q = 001$

to the set of breaking-patterns $\mathcal{BP}$. □

7.2 Dealing with super terms

First of all, it is worth to mention that super terms are more expressive than terms, that is, there are sets of messages that can be described as super terms but not as terms. This is for instance the case for the set of messages that contain the constant A as a sub-message. In fact, introducing the interpreted function symbol *Sup* corresponds to adding the sub-term relation to a logic on terms. Moreover, it is not difficult to exhibit examples of protocols where one needs the expressive power of super terms to represent the safe-breakers.

Unification and matching are the key operations in Step $(2a)$ of the symbolic algorithm. The problem we need to solve for obtaining our symbolic algorithm is, however, *not* the unification of super terms. The problem we need to solve is the following: Given two super terms u and t, we have to determine the set $\mathcal{U}(u,t)$ of substitutions σ such that there exist terms $u' \in [\![u]\!]$ and $t' \in [\![t]\!]$ such that $\sigma(u') = \sigma(t')$. More precisely, we want to characterize the set of most general unifiers that unify some terms in $[\![u]\!]$ and $[\![t]\!]$. Actually, the problem we need to solve for our symbolic algorithm is a simpler one where at least one of the super terms u and t is simply a term, that is, without occurence of *Sup* in it[6]. We prefer, however, to present a solution for the general case. We will do this in a general setting.

Let us consider a finite set $\mathcal{F}$ of function symbols such that *Sup* $\notin \mathcal{F}$ and let $\mathcal{X}$ be a countable set of variables (see Section 2 for the notations). The set of super terms induced by $\mathcal{F}$ and $\mathcal{X}$, denoted by $\mathcal{ST}(\mathcal{F},\mathcal{X})$ is defined by the following BNF:

$$t ::= x \mid f(t_1, \cdots, t_n) \mid Sup\,(t)$$

where x is a variable in $\mathcal{X}$ and $f \in \mathcal{F}$ is a function symbol of arity $n \geq 0$. Let $\mathcal{F}^{(i)}$ denote the function symbols in $\mathcal{F}$ of arity i. As usual, function symbols of arity 0, i.e. elements of $\mathcal{F}^{(0)}$, are called constants. The meaning $[\![t]\!]$ of a super term t is a set of terms in $\mathcal{T}(\mathcal{X},\mathcal{F})$, it has been defined in Definition 6.1.

Definition 7.1 *Given two super terms u and v, a substitution $\sigma : \mathcal{X} \to \mathcal{ST}(\mathcal{X},\mathcal{F})$ is called a* maximal general unifier *for u and v, if the following conditions are satisfied:*

1. *it is a most general unifier for some terms $t \in [\![u]\!]$ and $t' \in [\![v]\!]$ and*
2. *for every substitution σ' that unifies terms in $[\![u]\!]$ and $[\![v]\!]$, σ' is not more general than σ, that is, for no substitution ρ, we have $\sigma = \rho\sigma'$.*

We denote by $\mathcal{U}(u,v)$ the set of maximal general unifiers for u and v. □

In general there will be more than one maximal general unifier for u and v even modulo renaming. The definition of $\mathcal{U}$ can be extended in the usual way –as for unification– to sets $\{(u_i, v_i) \mid i \in \mathbb{N}_n\}$ of pairs of super terms. In the sequel, we prefer to write $u_i = v_i$ instead of (u_i, v_i) as our algorithm essentially consists in manipulating some kind of equations.

In this section, we want to develop an algorithm that given $E = \{u_i = v_i \mid i \in [1,n]\}$ determines $\mathcal{U}(E)$. From now on, we will call such a set E a *generalized equational problem*, written GEP for short. It turns out that an extension of the set of transformations that solve the usual unification problem (cf. [6]) will give the solution.

We recall in Figure 7 the usual six rules of [6] for solving unification and we add three rules to deal with the *Sup* operator.

We only solve the unification problem in the case of a signature $\mathcal{F}$ with at least a constructor of arity greater than one ; we do not present here the rules for the case of signature with only unary constructors which are useless in the context of crytography. We attract the reader's attention to the fact that the *Sup-Splitting* rule transforms a GEP E into a set of GEPs. Indeed, it yields a new GEP for each sub-term of $f(t_1, \cdots, t_n)$. This is not the case for the usual unification rules.

Example 7.2 Consider the following GEP

$$\begin{cases} [1] & Sup\,(x) = f(b, g(Sup\,(a))) \\ [2] & f(b, Sup\,(g(x))) = f(b, g(b)) \end{cases}$$

Equation 1 is eliminated by the rule (Sup-Delete-2) of Figure 7. Indeed, it is equivalent to $x \preceq f(b, g(Sup\,(a)))$ which puts no constraint on x as long as the term signature contains a contructor with arity greater than one (*e.g.*, **pair**). Indeed, whatever the term we obtain for x it is always possible, using binary

[6] Indeed, we need to unify the conclusion of a rule, which is a term, with a breaking-patterns which can be a super term.

We consider the case $\bigcup_{i\geq 2}\mathcal{F}^{(i)}\neq\emptyset$ with $f,g\in\mathcal{F}^{(n)}$; $x\in\mathcal{X}$; $u,v,u_i,v_i\in\mathcal{ST}$; $t,t_i\in\mathcal{T}$ and $Sup\notin\mathcal{F}$		
Delete	$\{u=u\}\cup E$	$\Rightarrow E$
Orient	$\{u=x\}\cup E$	$\Rightarrow \{x=u\}\cup E$, if $u\notin\mathcal{X}$
Decompose	$\{f(u_1,\cdots,u_n)=f(v_1,\cdots,v_n)\}\cup E$	$\Rightarrow \{u_i=v_i \mid i\in\mathbb{N}_n\}\cup E$
Clash	$\{f(u_1,\cdots,u_n)=g(v_1,\cdots,v_m)\}\cup E$	$\Rightarrow \perp$
Eliminate	$\{x=u\}\cup E$	$\Rightarrow \{x=u\}\cup E[u/x]$, if $x\notin var(u)$
Occurs-Check	$\{x=u\}\cup E$	$\Rightarrow \perp$, if $x\in var(u)\wedge u\neq Sup(x)\wedge u\neq x$
Sup-Delete-1	$\{x=Sup(x)\}\cup E$	$\Rightarrow E$
Sup-Delete-2	$\{Sup(u)=v\}\cup E$	$\Rightarrow E$, if $Sup(.)$ appears in v
Sup-Splitting	$\{Sup(u)=f(t_1,\cdots,t_n)\}\cup E$	$\Rightarrow \{u=t\}\cup E$, for $t\preceq f(t_1,\cdots,t_n)$, if $Sup(.)$ does not appear in $f(t_1,\cdots,t_n)$

Figure 7: Usual rules for solving unification extended to deal with superterms

contructors, to adjust the *Sup* part in $f(b,g(Sup(a)))$ in order to obtain a term that contains x. As an example, $f(b,g(\mathbf{pair}(x,a))$ contains x and it is an instance of $f(b,g(Sup(a)))$.

The rule *Decompose* removes Equation 2 and produces the constraints $Sup(g(x)))=g(b)$ and $b=b$ (which is eliminated by the rule *Delete*). Then, by rule *Sup-Splitting*, the former equation yields two GEPs: $\{g(x)=g(b)\}$ and $\{g(x)=b\}$. Finally, we obtain the solution $x=b$. □

Termination of the algorithm can be proved using lexicographic ordering and the ranking function that maps a GEP E to (m_1,m_2,m_3), where:

- m_1 is the number of variables in E that are not solved. As usual, a variable x is *solved* in E if it occurs exactly once in E, namely on the left-hand side of some equation $x=u$ with $x\notin var(u)$.
- m_2 is the measure of E defined by $\mathcal{M}(E)=\Sigma_{u=v\in E}(|u|+|v|)$ and $\mathcal{M}(\perp)=0$,
- m_3 is the number of equations $u=x$ in E with $x\in\mathcal{X}$ and $u\notin\mathcal{X}$.

The application of a rule to a GEP E leads to one or more GEPs with a lower rank than E. Although the *Sup-Splitting* rule of Figure 7 increases the number of GEPs, this number is bounded by the number of subterms of the right-hand side term. The ranking function and the bounded number of deriveable GEPs ensure the termination of the algorithm.

To prove soundness of the algorithm, we prove for each rule $E\Rightarrow E_1,\cdots,E_n$ that we have $\mathcal{U}(E)=\bigcup_{1\leq i\leq n}\mathcal{U}(E_i)$.

7.3 On the termination of the symbolic algorithm

In this section, we present a technique that makes a depth-first implementation of the symbolic verification algorithm terminate more often, at a price of a safe approximation of the results. In fact, our prototype implementation of our verification algorithm, named HERMES, terminates with precise results on all practical examples of protocols we tried. That is, the results did not show any false attack (see Table 8).

A sequence $(t_i,p_i)_{i\geq 0}$ of breaking-patterns is called *increasing at a sequence* $(q_i)_{i\geq 0}$ of positions, if the following conditions are satisfied for every $i\geq 0$:

1. $q_i\in dom(t_i)$ and $q_i\preceq q_{i+1}$,
2. $t_i[z/q_0]=t_0[z/q_0]$, where z is fresh variable.
3. $(t_{i|q_i},q_i^{-1}p_i)=(t_{0|q_0},q_0^{-1}p_0)$.

Let us consider an example to clarify these definitions.

Protocol Name	Result	Time (sec)
Yahalom	OK	12.67
Needham-Schroeder Public Key	Attack	0.04
Needham-Schroeder Public Key (with a key server)	Attack	0.90
Needham-Schroeder-Lowe	OK	0.03
Otway-Rees	OK[1]	0.01
Denning Sacco Key Distribution with Public Key	Attack	0.02
Wide Mouthed Frog (modified)	OK	0.04
Kao-Chow	OK	0.78
Neumann-Stubblebine	OK[1]	0.04
Needham-Schroeder Symmetric Key	Attack	0.08
ISO Symmetric Key One-Pass Unilateral Authentication	Attack	0.01
ISO Symmetric Key Two-Pass Unilateral Authentication	OK	0.01
Andrew Secure RPC	Attack	0.03

Figure 8: The results provided by HERMES, our prototype for verifying secrecy properties, running on a Pentium III 600Mhz PC under Linux 2.2.19.

Example 7.3 Consider the following rule from the session (A, A) of Needham-Schroeder-Lowe protocol presented in Section 7.4:

$$r = \{(A, (N_1^{AA}, y))\}_{K_A} \to \{y\}_{K_A}.$$

Consider the sequence $(\{\theta^i(I, x)\}_{K_A}, p_i)_{i \geq 0}$, where $\theta(z) = (A, (N_1^{AA}, z))$ and $p_i = 01 \cdot (11)^i$. The first three terms of the sequence are:
$(\{\theta^0(I, x)\}_{K_A} = \{(I, x)\}_{K_A}, 01)$
$(\{\theta^1(I, x)\}_{K_A} = \{(A, (N_1^{AA}, (I, x)))\}_{K_A}, 0111)$ and
$(\{\theta^2(I, x)\}_{K_A} = \{(A, (N_1^{AA}, (A, (N_1^{AA}, (I, x)))))\}_{K_A}$
$, 011111)$. The whole sequence can be obtained by iteratively computing the breaking-patterns induced by the rule r starting from the breaking-pattern $(\{(I, x)\}_{K_A}, 01)$. Thus, a naive application of our symbolic algorithm will not terminate. On the other hand, this sequence is increasing at $(q_i = 0 \cdot (11)^i)_{i \geq 0}$. Indeed, $\{\theta^i(I, x)\}_{K_A}[z/q_0] = \{z\}_{K_A}$ and
$((\{\theta^i(I, x)\}_{K_A})_{|q_i}, q_i^{-1} p_i = 1) = ((I, x), 1)$, for every $i \geq 0$. We will see now how this fact can be exploited to make the algorithm to converge. □

The idea of our technique for enforcing termination of the symbolic algorithm is expressed by the following proposition:

Proposition 7.1 *Let $(t_i, p_i)_{i \geq 0}$ be increasing at $(q_i)_{i \geq 0}$. Then,*

$$\begin{aligned} \bigcup_{i \geq 0} [\![(t_i, p_i)]\!] \subseteq \; & \bigcup_{i<j} [\![(t_i, p_i)]\!] \\ & \cup [\![t_j[\mathit{Sup}\,(t_{j_{|q_j}})/q_j], q_j \cdot 0 \cdot q_j^{-1} p_j]\!] \\ & \text{for every } j \geq 0. \end{aligned}$$

Example 7.4 Consider again our Example 7.3.
Then, if we choose $j = 1$, we obtain a set consisting of the two super terms $(\{(I, x)\}_{K_A}, 01)$ and $(\{(A, (N_1^{AA}, \mathit{Sup}\,(I, x)))\}_{K_A}, 011$
which approximates the whole sequence $\left(\{\theta^i(I, x)\}_{K_A}, p_i\right)_{i \geq 0}$.

[1]There is a known attack of the untyped version of the protocol. This attack relies on the misuse of a message as an encryption key. Discovering this type attack automatically requires to deal with non-atomic keys. This is not yet implemented in HERMES.

$$\boxed{\begin{array}{l}\{I, x_s\}_{K_A},\\ \{A, (N_1^{AA}, \mathit{Sup}\,(I, x_s))\}_{K_A},\\ \{A, (N_1, \mathit{Sup}\,(I, x_s))\}_{K_A}.\end{array}}$$

Figure 9: The breaking-patterns for the Needham-Schroeder-Lowe protocol

7.4 Needham-Schroeder-Lowe Protocol

The corrected version of the Needham-Schroeder protocol is also called Needham-Schroeder-Lowe as it is G. Lowe who found the attack and corrected the protocol. The difference with the initial version is in the second transition of principal B:

$$\begin{array}{lcl} A \to B & : & \{A, N_1\}_{K_B} \\ B \to A & : & \{B, N_1, N_2\}_{K_A} \\ A \to B & : & \{N_2\}_{K_B} \end{array}$$

In practice, we notice that if (pt, p) is a breaking-pattern then the pattern at the position p in pt is a variable. Therefore, for the sake of readability, further we will write only the pattern pt instead of the breaking-pattern (pt, p). The position is indicated by the subscript s to the variable that is at the position p in the pattern pt.

We run our verification algorithm with $\mathcal{S} = \{N_2, K_A^{-1}\}$, the empty set of breaking-patterns and the set of keys $K = \{K_A\}$. The algorithm terminates with the set of secrets unchanged and the set PB of breaking-patterns given in Figure 9. As the initial constraints are $E_0 \not\vdash^{\in_c} \{N_1, N_2, K_A^{-1}\}$, that is, none of the messages in $\{N_1, N_2, K_A^{-1}\}$ is contained at a critical position in a message derivable from E_0, it is easy to prove that we have $E_0 \langle PB \rangle_K \mathcal{S}$. Hence, we can conclude that the Needham-Schroeder-Lowe protocol preserves the secret N_2. Concerning, the uncorrected version of Example 3.1, during computation of new secrets and breaking-patterns, we arrive at a situation where we have to add $\{A, N_1^{AI}\}_{K_I}$ as a secret. As this message contains neither a fresh nonce nor a secret, we stop the computation and follow it back to try constructing an attack. This way, we obtain the attack known as "man in the middle". □

8 Conclusion

In this paper, we presented a method based on abstract interpretation for verifying secrecy properties of cryptographic protocols in a general model. Our method deals with unbounded number of sessions, unbounded number of principals, unbounded message depth and unbounded creation of fresh nonces. However, in contrast to the work in [5, 35, 30], where the session number is bounded, our method is not complete. Indeed, the problem is in its most general form undecidable even when pairing is not allowed as shown in [4]. The main contribution of the paper is a verification algorithm that consists of computing an inductive invariant using super as symbolic representation. Our method can already deal with models in which we distinguish between long term and short term keys and which contain variables ranging over keys. The idea here is that short term keys can be revealed to the intruder when a session has terminated. This is not the case for long term keys. This allows a more faithful modeling of some protocols.

An version of our tool together with the examples of Table 8 is available at the url: `http://www-verimag.imag.fr/~lbozga/hermes/hermes.php`.

References

[1] M. Abadi. Secrecy by typing in security protocols. In *Theoretical Aspects of Computer Software*, volume 1281 of *LNCS*, pages 611–638, 1997. 1

[2] M. Abadi and B. Blanchet. Secrecy Types for Asymmetric Communication. In *Foundations of Software Science and Computation Structures*, volume 2030 of *LNCS*, pages 25–41, 2001. 1

[3] M. Abadi and B. Blanchet. Analyzing Security Protocols with Secrecy Types and Logic Programs. In *ACM Symposium on Principles of Programming Languages*, pages 33–44, 2002. 1

[4] R. M. Amadio and W. Charatonik. On name generation and set-based analysis in the Dolev-Yao model. In *CONCUR: 13th International Conference on Concurrency Theory*. LNCS, Springer-Verlag, 2002. 8

[5] R. M. Amadio and D. Lugiez. On the reachability problem in cryptographic protocols. In *International Conference on Concurrency Theory*, volume 1877 of *LNCS*, pages 380–394, 2000. 1, 1, 8

[6] F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998. 7.2

[7] B. Blanchet. Abstracting cryptographic protocols by prolog rules. In *International Static Analysis Symposium*, volume 2126 of *LNCS*, pages 433–436, 2001. 1

[8] D. Bolignano. An approach to the formal verification of cryptographic protocols. In *ACM Conference on Computer and Communications Security*, pages 106–118, 1996. 3.3.1

[9] D. Bolignano. Integrating proof-based and model-checking techniques for the formal verification of cryptographic protocols. In *International Computer Aided Verification Conference*, volume 1427 of *LNCS*, pages 77–87, 1998. 1

[10] L. Bozga, C. Ene, and Y. Lakhnech. On the existence of an effective and complete proof system for bounded security protocols. In *FOSSACS*, volume 2987. lncs, 2004. 1

[11] L. Bozga, C. Ene, and Y. Lakhnech. A symbolic decision procedure for bounded security protocols with time stamps. Technical report, Verimag, 2004. 1

[12] John A. Clark and Jeremy L. Jacob. A survey of authentication protocol literature. Version 1.0, University of York, Department of Computer Science, November 1997. 1, 1, C

[13] E.M. Clarke, S. Jha, and W. Marrero. Using state space exploration and a natural deduction style message derivation engine to verify security protocols. In *IFIP Working Conference on Programming Concepts and Methods*, 1998. 1

[14] H. Comon, V. Cortier, and J. Mitchell. Tree automata with one memory, set constraints, and ping-pong protocols. In *International Colloquium on Automata, Languages and Programming*, volume 2076 of *LNCS*, 2001. 1

[15] H. Comon and V. Shmatikov. Is it possible to decide whether a cryptographic protocol is secure or not? *Journal of Telecommunications and Information Technology*, 2002. 1

[16] H. Comon-Lundh. and V. Cortier. Security properties: Two agents are sufficient. *Science of Computer Programming*, 50(1-3):51 – 71, March 2004. Special issue on 12th European symposium on programming (ESOP 2003). 4

[17] V. Cortier, J. Millen, and H. Rueß. Proving secrecy is easy enough. In *IEEE Computer Security Foundations Workshop*, pages 97–110, 2001. 1

[18] D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983. 3.3

[19] N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. Undecidability of bounded security protocols. In *Workshop on Formal Methods and Security Protocols*, 1999. 1, 1

[20] S. Even and O. Goldreich. On the security of multi-party ping pong protocols. Technical report, Israel Institute of Technology, 1983. 1

[21] F.J.T. Fábrega, J.C. Herzog, and J.D. Guttman. Strand Spaces: Why is a Security Protocol Correct ? In *IEEE Conference on Security and Privacy*, pages 160–171, 1998. 3.3

[22] T. Genet and F. Klay. Rewriting for cryptographic protocol verification. In *International Conference on Automated Deduction*, volume 1831 of *LNCS*, 2000. 1, 1

[23] A. Gordon and A. Jeffrey. Authenticity by typing for security protocols. In *IEEE Computer Security Foundations Workshop*, pages 145–159, 2001. 1

[24] Jean Goubault-Larrecq. A method for automatic cryptographic protocol verification. In *International Workshop on Formal Methods for Parallel Programming: Theory and Applications*, volume 1800 of *LNCS*, 2000. 1, 1

[25] Nabil El Khadi. Automatic verification of confidentiality properties of cryptographic program. *Network Information System Journa*, 6, 2001. 1

[26] G. Lowe. An attack on the Needham-Schroeder public-key authentification protocol. *Information Processing Letters*, 56(3):131–133, 1995. 1

[27] G. Lowe. Breaking and fixing the Needham-Schroeder Public-Key protocol using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1055 of *LNCS*, pages 147–166, 1996. 1

[28] G. Lowe. Casper: A compiler for the analysis of security protocols. In *10th IEEE Computer Security Foundations Workshop (CSFW '97)*, pages 18–30, Washington - Brussels - Tokyo, June 1997. IEEE. 1

[29] C. Meadows. Invariant generation techniques in cryptographic protocol analysis. In *Computer Security Foundations Workshop*, 2000. 1

[30] J. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *ACM Conference on Computer and Communications Security*, pages 166–175, 2001. 1, 1, 8

[31] J. C. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using murφ. In *Conference on Security and Privacy*, pages 141–153, 1997. 1

[32] D. Monniaux. Abstracting Cryptographic Protocols with Tree Automata. In *Static Analysis Symposium*, volume 1694 of *LNCS*, pages 149–163, 1999. 1, 1

[33] R.M. Needham and M.D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978. 1

[34] L. Paulson. Proving properties of security protocols by induction. In *IEEE Computer Security Foundations Workshop*, pages 70–83, 1997. 1

[35] M. Rusinowitch and M. Turuani. Protocol insecurity with finite number of sessions is NP-complete. In *IEEE Computer Security Foundations Workshop*, 2001. 1, 1, 8

[36] J. Thayer, J. Herzog, and J. Guttman. Honest Ideals on Strand Spaces. In *IEEE Computer Security Foundations Workshop*, pages 66–78, 1998. 3.3

[37] Christoph Weidenbach. Towards an Automatic Analysis of Security Protocols in First-Order Logic. In *International Conference on Automated Deduction*, volume 1632 of *LNCS*, pages 314–328, 1999. 1

A Definitions

Definition A.1 (K-guards) *Let K be a set of keys. The K-*guards *are messages of the form $\{m\}_k$ for some $m \in \mathcal{T}(\mathcal{F})$ and $k \in K$.*

$$K\text{-}guards = \big\{\{m\}_k \mid m \in \mathcal{T}(\mathcal{F}),\ k \in K\big\}$$

Definition A.2 (least protecting position) *Let t be any term and p be a position. The* least p-protecting position of p in t*, denoted by* lpp(t, p)*, is the position of the highest K-guard protecting position p of t. Formally,*

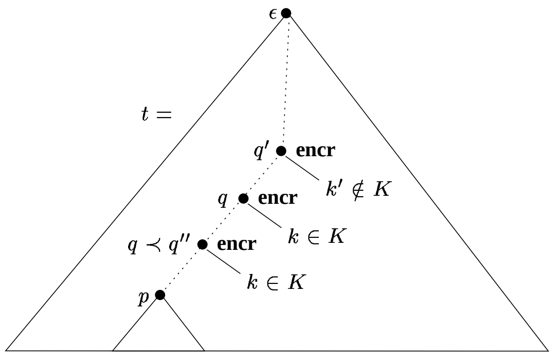

Figure 10: Position q is the least p-protecting position in term t

$$lpp(t, p) = \\ min\big(\ \{q \mid q \prec p,\ t_{|q} \text{ is a } K\text{-guard}\}\ \big)$$

This definition is illustrated in Fig. 10.

Definition A.3 (sub-safe-breakers) *Let (b, p) be a safe-breaker. Then, $ssb(b, p)$ denotes the* sub-safe-breakers *of (b, p), that is the set of all* proper sub-terms of b *that are safe-breakers for position p. The sub-safe-breakers of (b, p) are built from the K-guards of b which are above the position p.*

$$ssb(b, p) = \\ \big\{(b_{|q}, p') \mid q \cdot p' = p,\ b_{|q} \text{ is a } K\text{-guard}\big\} \\ - \{(b, p)\}$$

B Proofs

B.1 Proof of proposition 5.1

Proposition 5.1 Let E be a set of messages and $(\mathcal{B}, \mathcal{S})$ be a pair of safe-breakers and secrets. If $(\mathcal{B}, \mathcal{S})$ is *well-formed* and $E\langle\mathcal{B}\rangle_K\mathcal{S}$ holds, then the secrets of $\mathcal{S}$ are insensitive to $\mathcal{B}$ in any message m derivable from E, that is, $E \vdash m \Rightarrow m\langle\mathcal{B}\rangle_K\mathcal{S}$.

Proof Before tackling the proof, we introduce the following definition:

We say that m is a *derivation-minimal counter-example*, if the following conditions are satisfied:

1. $E \vdash m$,
2. $\neg m\langle\mathcal{B}\rangle_K\mathcal{S}$ and
3. there is a derivation for $E \vdash m$ which does not contain any strict sub-derivation $E \vdash m'$ of a message m' with $\neg m'\langle\mathcal{B}\rangle_K\mathcal{S}$.

Assume that $E \vdash s$ for an $s \in \mathcal{S}$. Then, there exists a derivation-minimal counter-example m such that $\neg m\langle\mathcal{B}\rangle_K\mathcal{S}$. The existence of m can be proved as follows. Take a derivation of $E \vdash m$ and let N_0 be its size. If m is not a derivation-minimal counter-example then there must exist a sub-derivation $E \vdash m'$ with $\neg m'\langle\mathcal{B}\rangle_K\mathcal{S}$. Clearly, the size N_1 of the derivation tree of m' is strictly smaller than N_0. Repeated application of the same argument must lead to a derivation-minimal counter-example as there are no strictly decreasing chains in $\mathbb{N}$.

We come back to the proof of Proposition 5.1. Let us assume that the pair $(\mathcal{B}, \mathcal{S})$ is well-formed, that $E\langle\mathcal{B}\rangle_K\mathcal{S}$ holds. Moreover assume that there exists a message m derivable from E, which is derivation-minimal counter-example, meaning that $\neg\big(m\langle\mathcal{B}\rangle_K\, s\big)$ for a secret $s \in \mathcal{S}$. Then, we derive a contradiction by case analysis on the last derivation step in $E \vdash m$.

1. If the last step is an application of the rule $m \in E \Rightarrow E \vdash m$. This contradicts the assumption $E\langle\mathcal{B}\rangle_K\mathcal{S}$, since $m \in E$ and $\neg m\langle\mathcal{B}\rangle_K\, s$.

2. Consider the case of encryption with a key k from K, that is, $m = \{m'\}_k$ and the last step is an application of the rule $E \vdash m' \wedge E \vdash k \Rightarrow E \vdash \{m'\}_k$. We know that $m' \langle\mathcal{B}\rangle_K\, s$ (1) and $k\ \langle\mathcal{B}\rangle_K\, s$ (2) from the fact that m is a derivation-minimal counter-example. Our hypothesis that should lead to contradiction becomes $\neg\{m'\}_k \langle\mathcal{B}\rangle_K\, s$. According to Definition 5.1, in the case where $k \in K$, two rules can lead to the conclusion $\neg\{m'\}_k \langle\mathcal{B}\rangle_K\, s$.

   The first rule corresponds to the case where $s = \{m'\}_k$. Then m' or k belongs to $\mathcal{S}$ since the set of secrets $\mathcal{S}$ is closed against composition. So, m' or k is a derivation counter-example smaller than m, the minimal one: contradiction.

   The conclusion $\neg\{m'\}_k \langle\mathcal{B}\rangle_K\, s$ can also results from an application of the last rule of Definition 5.1 for a position p in $\{m'\}_k$:
$$\frac{k \in K,\ \neg(\{m'\}_k)_{|_p} \langle\mathcal{B}\rangle_K\, s,\ (\{m'\}_k, p) \in \mathcal{B}}{\neg\{m'\}_k \langle\mathcal{B}\rangle_K\, s}$$
   Then, the important facts for the discussion are (3) $(\{m'\}_k, p) \in \mathcal{B}$ and (4) $\neg(\{m'\}_k)_{|_p} \langle\mathcal{B}\rangle_K\, s$. Again, we have to consider two cases:

   - If $ssb(\{m'\}_k, p) = \emptyset$ then the only K-guard protecting position p is $\{m'\}_k$. So, the secret at position p in $\{m'\}_k$ is protected neither in m', nor in k. This contradicts the facts (1) $m' \langle\mathcal{B}\rangle_K\, s$ and (2) $k\ \langle\mathcal{B}\rangle_K\, s$ given by the minimality of m.
   - In the case $ssb(\{m'\}_k, p) \neq \emptyset$, we come to the same contradiction. Let (b', p') be the greatest element of $ssb(\{m'\}_k, p)$, meaning that (5) there is no K-guard above b' in $\{m'\}_k$. By definition of ssb $(\{m'\}_k, p)$, we know that b' is a proper sub-term of $\{m'\}_k$; it is a K-guard and $b'_{|_{p'}} = \left(\{m'\}_k\right)_{|_p}$. So, (3) $\neg(\{m'\}_k)_{|_p} \langle\mathcal{B}\rangle_K\, s$ entails (6) $\neg b'_{|_{p'}} \langle\mathcal{B}\rangle_K\, s$. Additionally, we know that (7) $(b', p') \in \mathcal{B}$ since (3) $(\{m'\}_k, p) \in \mathcal{B}$, $(\mathcal{B}, \mathcal{S})$ is well-formed and $b' \prec \{m'\}_k$. Then, an application the last rule of Definition 5.1 (for K-guards) to (6) and (7) yields $\neg b' \langle\mathcal{B}\rangle_K\, s$.
     We can assume that b' is a subterm of m' (the same reasoning works for the case $b' \preceq k$; we then obtain a contradiction between $\neg k \langle\mathcal{B}\rangle_K\, s$ and (2)). By (5), the maximality of b', there is no K-guard protecting b' in m'. Then, we can deduce $\neg m' \langle\mathcal{B}\rangle_K\, s$ from $\neg b' \langle\mathcal{B}\rangle_K\, s$ using rules of Definition 5.1: this contradicts fact (1) $m' \langle\mathcal{B}\rangle_K\, s$.

3. If the last step is an encryption with a key k which is not in K, that is, $m = \{m'\}_k$ and $E \vdash m' \wedge E \vdash k \Rightarrow E \vdash \{m'\}_k$. The argumentation is similar to the one used for the previous item. In particular, the fact (1) holds. Since m is a derivation-minimal counter-example, the judgment $\neg m \langle\mathcal{B}\rangle_K\, \mathcal{S}$ holds and comes as a conclusion of the first rule or the second rule of Definition 5.1. The case of the first rule is treated as in the previous item. The case of the second rule, $\frac{\neg m' \langle\mathcal{B}\rangle_K\, s,\ k \notin K}{\neg\{m'\}_k \langle\mathcal{B}\rangle_K\, s}$, requires $\neg m' \langle\mathcal{B}\rangle_K\, s$ which contradicts fact (1) $m' \langle\mathcal{B}\rangle_K\, s$.

4. The case of pairing is very similar to the previous case.

5. The case of projection also contradicts the derivation-minimality assumption.

6. If the last step is a decryption with a key k^{-1}, that is, $E \vdash \{m\}_k \wedge E \vdash k^{-1} \Rightarrow E \vdash m$ and we assumed that $\neg m \langle\mathcal{B}\rangle_K\, s$. We condiser two cases: If $k \notin K$ then we obtain $\neg\{m\}_k \langle\mathcal{B}\rangle_K\, s$ by the second rule of Definition 5.1. So, $\{m\}_k$ is a derivation counter-example smaller than m: this contradicts the derivation-minimality of m. On the other hand, it $k \in K$ then k^{-1} belongs to S as a consequence of the well-formedness of $(\mathcal{B}, \mathcal{S})$ w.r.t. K. So, k^{-1} is a derivation counter-example smaller than m: contradiction.

□

B.2 Proof of theorem 5.1

Theorem 5.1 Let $\mathcal{S}$ be a set of secrets and $\mathcal{B}$ be a set of safe-breakers. If $(\mathcal{B}, \mathcal{S})$ is well-formed and stable w.r.t. all rules in $\mathcal{R}$; if additionally $E_0 \langle\mathcal{B}\rangle_K\, \mathcal{S}$ holds for every set of messages E_0 that satisfies $\mathcal{C}$, then $\not\models_P \mathcal{S}$, i.e., the secrets in $\mathcal{S}$ are preserved in any execution of the protocol $P = (\mathcal{C}, \mathcal{R})$.

Proof We proof by induction that for any run $E_0 \xrightarrow{r_1} E_1 \cdots E_{n-1} \xrightarrow{r_n} E_n$, where for each $i = 1, \cdots n$, there is a substitution $\rho_i : \mathcal{X} \to \mathcal{T}(\mathcal{F})$ such that $E_{i-1} \vdash \rho(t_1)$ and $E_i = E_{i-1} \cup \{\rho(t_2)\}$, where $t_1 \to t_2 = r_i$, we have $E_n \not\vdash \mathcal{S}$.

$$
\begin{array}{lcl}
A \to B & : & A, N_1 \\
B \to S & : & B, \{A, N_1, N_2\}_{k_{BS}} \\
S \to A & : & \{B, k_{AB}, N_1, N_2\}_{k_{AS}}, \{A, k_{AB}\}_{k_{BS}} \\
A \to B & : & \{A, k_{AB}\}_{k_{BS}}, \{N_2\}_{k_{AB}}
\end{array}
$$

Figure 11: The Yahalom protocol

$$
\begin{array}{lrcl}
\mathit{tran}(p_1): & p_1 & \to & p_1, n_1 \\
\multicolumn{2}{r}{\{p_2, smk(m, x_1, y_1), n_1, z_1\}_{smk(p_1,p_3)}, W_1} & \to & W_1, \{z_1\}_{smk(m,x_1,y_1)} \\
\mathit{tran}(p_2): & x_2, y_2 & \to & p_2, \{x_2, y_2, n_2\}_{smk(x_2,p_3)} \\
\mathit{tran}(p_3): & x_3, \{y_3, z_3, W_3\}_{smk(x_3,p_3)} & \to & \{x_3, smk(n_3, y_3, x_3), z_3, W_3\}_{smk(y_3,p_3)}, \\
 & & & \{y_3, smk(n_3, y_3, x_3)\}_{smk(x_3,p_3)}
\end{array}
$$

Figure 12: The Yahalom protocol transitions

First, we have $E_0 \langle \mathcal{B} \rangle_K \mathcal{S}$ then $E_0 \not\vdash \mathcal{S}$.

Second, we proof that if for any run we have $E_{i-1} \langle \mathcal{B} \rangle_K \mathcal{S}$ then, we have $E_i \langle \mathcal{B} \rangle_K \mathcal{S}$, for all rules $r = t_1 \to t_2$ in $\mathcal{R}$ and for all ρ such that $E_{i-1} \vdash \rho(t_1)$ and $E_i = E_{i-1} \cup \{\rho(t_2)\}$.

We have $E_{i-1} \langle \mathcal{B} \rangle_K \mathcal{S}$ and $E_{i-1} \vdash \rho(t_1)$ so we are in the hypothesis of the proposition 5.1 then $\rho(t_1) \langle \mathcal{B} \rangle_K \mathcal{S}$. $(\mathcal{B}, \mathcal{S})$ is stable w.r.t. all rules in $\mathcal{R}$ then $\rho(t_2) \langle \mathcal{B} \rangle_K \mathcal{S}$. So we have $E_i \langle \mathcal{B} \rangle_K \mathcal{S}$ $\square$

C Example: The Yahalom Protocol

The aim of the Yahalom protocol (cf. [12] and see Figure 11) is to establish a secret symmetric shared key k_{AB} between two participants A and B using a trusted server S. The protocol assumes that A and B already share secure keys k_{AS}, respectively k_{BS} with the server S. The Yahalom protocol can be represented in our setting as follows: $P = \{p_1, p_2, p_3\}$ with *fresh*$(p_1) = \{n_1\}$, *fresh*$(p_2) = \{n_2\}$ and *fresh*$(p_3) = \{n_3\}$. The transitions are described in Figure 12.

The abstraction defined in Section 4 yields the following abstract sets:

$$
\begin{array}{l}
P^\sharp = \{A, I\}, \\
K^\sharp = \{K_I, smk(A, A), K_{AB}, K_{AB}^{AAA}\}, \\
N^\sharp = \{N_1, N_1^{AAA}, N_1^{AIA}, N_2, N_2^{AAA}, N_2^{IAA}, N_I\}\}
\end{array}
$$

For the sake of conciseness we write K_{AB} instead of $smk(N_3, A, A)$ respectively K_{AB}^{AAA} instead of $smk(N_3^{AA}, A, A)$. Figure 13 presents only some abstract rules of the protocol. We run our verification algorithm on the whole set of abstract rules R, the set of secrets $S = \{N_2, K_{AB}, smk(A, A)\}$, the empty set of breaking-patterns, and the set of keys $K = \{smk(A, A), K_{AB}\}$.

The algorithm terminates with the set of secrets unchanged and a set of breaking-patterns $\mathcal{B}$ which, for lack of space, is not presented here. We assuming that none of the secrets appear at a critical position in a message derivable from the initial knowledge E_0 of the intruder. Formally,

$$E_0 \vdash m \Rightarrow N_2 \notin_c m \wedge K_{AB} \notin_c m \wedge smk(A, A) \notin_c m$$

Then, it is easy to prove that the initial knowledge of the intruder, E_0, has the property $E_0 \langle \mathcal{B} \rangle_K \mathcal{S}$. This is a sufficient condition to ensure that the Yahalom protocol preserves the set of secrets S.

$$
\begin{array}{lrl}
tran(p_1): & & \\
\pi = (A, A, A) & \{A, K_I, N_2^{AAA}, z_1\}_{smk(A,A)}, W_1 & \rightarrow W_1, \{z_1\}_{K_I}; \\
\pi = (A, I, A) & \{I, K, N_2^{AAA}, z_1\}_{smk(A,A)}, W_1 & \rightarrow W_1, \{z_1\}_K, K \in K^\sharp \\
tran(p_2): & & \\
\pi = (A, I, A) & I, y_2 & \rightarrow A, \{I, y_2, N_1^{IAA}\}_{smk(A,A)}; \\
\pi = (A, A, A) & A, y_2 & \rightarrow A, \{A, y_2, N_1^{AAA}\}_{smk(A,A)}; \\
tran(p_3): & & \\
\quad \pi = (x_3, y_3, A), & x_3, \{y_3, z_3, W_3\}_{smk(x_3,A)} & \rightarrow \quad \{x_3, K_{y_3 x_3}^{y_3 x_3 A}, z_3, W_3\}_{smk(y_3,A)}, \\
\quad \textit{with } x_3, y_3 \in P^\sharp & & \quad\quad \{y_3, K_{y_3 x_3}^{y_3 x_3 A}\}_{smk(x_3,A)};
\end{array}
$$

Figure 13: Some examples of abstract rules of the Yahalom protocol