Pattern-based Abstraction for Verifying Secrecy in Protocols *
Category 1

L. Bozga Y. Lakhnech † M. Périn

Verimag
2 av. de Vignate 38610 Gieres, France.
{lbozga,lakhnech,perin}@imag.fr

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 essentially use sets of so-called *pattern terms*. Pattern 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 (13.81 sec), Otway-Rees (0.01 sec), Skeme (0.06 sec) and Kao-Chow (0.92 sec)..

*This work has been partially sopported by the RNTL project EVA.
†Contact author

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 authentification. It has been introduced in 1978 [23]. An attack on this protocol has been found by G. Lowe using the CSP model-checker FDR in 1995 [17]; and this led to a corrected version of the protocol [18]. Consequently there has been a growing interest in developing and applying formal methods for validating cryptographic protocols [19, 10]. 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 capacity to forge new messages. It is known that reachability is undecidable for cryptographic protocols in the general case [14], even when a bound is put on the size of messages [13]. 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 [2, 26, 20] or to develop correct but incomplete verification algorithms as in [22, 16, 15].

In this paper, we present a correct but, in general, incomplete 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 an original method for proving that a secret is not revealed by a set of rules that model how the initial set of messages known by the intruder evolves. In contrast to almost all existing methods, we do not try to compute or approximate the sets of messages that can be known by the intruder. Our algorithm is rather based on the notion of "the secret being *guarded*, or *kept under the hat* of a safe message". 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. Then, any message that contains N_B and that is encrypted with K_B is a guard for N_B. For instance, N_B is guarded in the message $\{\{N_A, N_B\}_{K_B}\}_{K_I}$ by the sub-message $\{N_A, N_B\}_{K_B}$. The idea is then to compute a set of guards that will keep the secret unrevealed in all sent messages and such that the inverses of the keys used in this set are also secrets. The difficulty here is that this set is, in general, infinite. Therefore, we use *terms* to represent sets of guards. For instance the term $\{x, x_s\}_{K_B}$ says that the secret will be guarded in any message $\{m, m'\}_{K_B}$, where the secret is not a sub-message of m but may be a sub-message of m'. The problem is, however, that there might be a rule $\{I, y\}_{K_B} \to y$ that will send y unencrypted to the intruder if (s)he produces the message $\{I, y\}_{K_B}$. Hence, the term $\{x, x_s\}_{K_B}$ will guard the secret except when x is I. Thus, our abstract domain consists of pairs $(\mathcal{G}, \mathcal{B})$ of terms. Those in $\mathcal{G}$ correspond to good messages that guard the secrets, whereas those in $\mathcal{B}$ denote "bad exceptions", that is, the particular instances of terms in $\mathcal{G}$ that do not guard the secrets. A weakness of terms 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 *pattern terms*, that it, terms with an interpreted constructor, *Sup*, where a term *Sup* (t) is meant for the set of terms that contain t as sub-term. To use these pattern terms in our verification method, we solve a generalized form of the unification problem and introduce a widening operator.

We developed a prototype in Caml that implements this method. We have been able to verify several protocols taken from [7] including, for instance, Needham-Schroeder-Lowe (0.03 sec), Yahalom (13.81 sec), Otway-Rees (0.01 sec), Skeme (0.06 sec) and Kao-Chow (0.92 sec).

Related work 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 [7] falls in this class. Recently, Comon, Cortier and Mitchell [9] 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 [2, 26, 20] or when nonce creation is not allowed and the size of messages is bounded [13]. For the general case, on one hand model-checking tools have been applied to discover flaws in cryptographic protocols [17, 27, 21, 8]. The tool described in [8] is a model-cheker dedicated to cryptograhic protocols. On the other hand, methods based on induction and theorem proving have been developed (e.g. [24, 4, 11]). Closest to our work are partial algorithms based on abstract interpretation and tree automata that have been presented in [22, 16, 15]. 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.

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 finite subset of ω^* to $\mathcal{X} \cup \mathcal{F}$ such that $tr(u) \in \mathcal{F}^n$ iff $u \cdot j \in dom(tr)$, for every $j \in \{0, \cdots, n-1\}$. We identify terms with trees by associating to each term t a tree $Tr(t)$ as follows: (1) if x is a variable, then $dom(Tr(x)) = \{\epsilon\}$ and $Tr(x)(\epsilon) = x$, (2) if f is a constant symbol, then $dom(Tr(f)) = \{\epsilon\}$ and $Tr(f)(\epsilon) = f$ and (3) for a term $t = f(t_0, \cdots, t_{n-1})$, $dom(Tr(t)) = \{\epsilon\} \cup \bigcup_{i=0}^{n-1} i \cdot dom(Tr(t_i))$, where $\cdot$ is word concatenation extended to sets, $Tr(t)(\epsilon) = f$ 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 $x \cdot p \in dom(t)$. 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 $var(t)$.

2 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.

2.1 Messages

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

1. *Principle names* are used to refer to the principles in a protocol. The set of all principles 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. We have the following atomic keys for each $p_1, \cdots, p_r \in \mathcal{P}^r$ where *pbk*, *pvk* and *smk* stand respectively for *public*, *private* and *symmetric* keys:

$$pbk(p_1, \cdots, p_r) \mid pvk(p_1, \cdots, p_r) \mid smk(p_1, \cdots, p_r).$$

   We denote by $\mathcal{AK}(p_1, \cdots, p_r)$ this set of keys and let $\mathcal{K} = \bigcup_{\vec{p} \in \mathcal{P}^+} \mathcal{AK}(\vec{p})$ denote the set of all keys. The key $pbk(p_1, \cdots, p_r)$ is an inverse of the key $pvk(p_1, \cdots, p_r)$ and vice versa; and a key $smk(p_1, \cdots, p_r)$ is its self-inverse. If k is a key then we use k^{-1} to denote its inverse. Moreover, as usual, we write K_A instead of $pbk(A)$, K_A^{-1} instead of $pvk(A)$ and K_{AB} instead of $smk(A, B)$.

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**. Thus, *messages* are ground terms in $\mathcal{T}(\mathcal{X}, \mathcal{F})$.

2.2 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 [12], 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 *introduction* rules while projection and decryption are called *elimination* rules. As usual, derivations in the system $\vdash$ can be seen as proof trees. Moreover, $\vdash$ can be seen as a natural deduction system. It enjoys the *normalized proof property* [25, 8] that states that if m is derivable from E then there is a proof of m from E where no elimination rule appears in the proof tree above (as a father of) any introduction rule.

For a term t, we use the notation $E \not\vdash t$ to denote that no instantiation 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 now define define *critical* and *non-critical* positions in a message. The idea is that since there is no way to deduce from an encrypted message the key with which it has been encrypted, the key position in messages of the form $\mathbf{encr}(m, k)$ is not critical. Formally, given a term t, a position p in t is called *non-critical*, if there is a position q such that $p = q \cdot 2$ and $t(q) = \mathbf{encr}$; 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 $m_{|p} = s$.

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

2.3 Representation of the protocol itself

We use in this paper a simple protocol representation that can be proved to be a safe approximation of a more realistic model [6]. The main idea behind this representation is the following abstraction. First, we fix an arbitrary session where the same principal, say A, plays the (say two) different roles in the protocol. Then, we identify the intruder I with all principles other than A. Moreover, we identify all sessions in which A is not involved. Concerning the other sessions, that is, those where A is involved, we identify:

- all sessions where A plays both roles and which are different from the fixed session,
- all sessions where A plays the first role while the second role is played by an other principal,
- all sessions where A plays the second role while the first role is played by a different principal.

Identifying sessions means also identifying the nonces and keys used in these sessions. This leaves us with a system where we have unbounded number of messages as the size of the messages is not bounded. To summarize, we model a protocol as a set of transitions that can be taken in any order and any number of times. One intriguing step in this abstraction is to take the same principal in all roles for the fixed session. As we show in [6], this abstraction covers all behaviours when distinct principals are associated to different roles. Informally, the reason is that although a principal can play two roles A and B, the knowledge he can get by playing each role are kept separate. We could consider different principals ; our method does not depend on this.

Thus, a protocol is represented by a pair $(\mathcal{C}, \mathcal{R})$ consisting of:

- a predicate $\mathcal{C}$ on sets of messages describing the messages that the intruder may initially know and
- a finite set $\mathcal{R}$ of rule schemata of the form $t_1 \to t_2$, where the t_i's are terms.

The constraint $\mathcal{C}$ can be used to describe freshness properties. For instance, if N is a nonce then the condition $E \not\vdash^{\in_c} N$ states that N is fresh with respect to E. In a companion paper [6], we describe how an abstract procotol description $(\mathcal{C}, \mathcal{R})$ is derived from a concrete one by abstract interpretation.

Let us turn our attention to the semantics of a protocol $(\mathcal{C}, \mathcal{R})$. A *run* of $(\mathcal{C}, \mathcal{R})$ is given by a fnite sequence of sets of messages E_i of the form

$$E_0 \xrightarrow{r_1} E_1 \cdots E_{n-1} \xrightarrow{r_n} E_n$$

with $n \geq 0$ such that E_0 satisfies the constraint $\mathcal{C}$ and such that, 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 $t_1 \to t_2 = r_i$.

In other words, considering E_{i-1} as the current knowledge of the intruder, a rule $t_1 \to t_2 = r_i$ can be taken if there is an instance $\rho(t_1)$ of t_1 that is derivable from E_{i-1}. The effect of applying the rule culminates in adding the message $\rho(t_2)$ to the knowledge of the intruder. Notice how following Bolignano [5] communication is modeled through the knowledge of the intruder. That is, the intruder can intercept messages use them to create fake messages and deliver these to the principals.

2.4 Secrecy modeling

Our goal is to determine whether secrecy property holds: for instance whether the intruder can get a secret s. That is, does a run exist that leads to a set E of messages from which s is derivable.

A *secret* is given by a message m. A protocol P satisfies the secrecy property defined by m, denoted by $\not\vdash_P m$, if it does not admit any run $E_0 \xrightarrow{r_1} E_1 \cdots E_{n-1} \xrightarrow{r_n} E_n$ such that $E_n \vdash m$. The definition of secrecy can be pointwisely extended to a set of secrets.

Example 2.1 The Needham-Schroeder protocol for authentification can be described as follows using the usual informal notation for cryptographic protocols:

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

Intuitively, A plays the role of the initiator of the session; while B is a responder.

In our model which yields an over-approximation of the possible runs of the protocol, we can describe the Needham-Schroeder protocol by the following rules. We write $\frac{t_1}{t_2}$ instead of $t_1 \to t_2$.

session (A, I)	session (I, A)	any session (A, A)	the fixed session (A, A)
$\frac{-}{\{A, N_1^{AI}\}_{pbk(I)}}$;	$\frac{-}{\{I, N_I\}_{pbk(A)}}$;	$\frac{-}{\{A, N_1^{AA}\}_{pbk(A)}}$;	$\frac{-}{\{A, N_1\}_{pbk(A)}}$;
$\frac{\{A, y\}_{pbk(I)}}{\{y, N_I\}_{pbk(A)}}$;	$\frac{\{I, y\}_{pbk(A)}}{\{y, N_2^{IA}\}_{pbk(I)}}$;	$\frac{\{A, y\}_{pbk(A)}}{\{y, N_2^{AA}\}_{pbk(A)}}$;	$\frac{\{A, y\}_{pbk(A)}}{\{y, N_2\}_{pbk(A)}}$;
$\frac{\{N_1^{AI}, z\}_{pbk(A)}}{\{z\}_{pbk(I)}}$;	$\frac{\{N_I, z\}_{pbk(I)}}{\{z\}_{pbk(A)}}$;	$\frac{\{N_1^{AA}, z\}_{pbk(A)}}{\{z\}_{pbk(A)}}$;	$\frac{\{N_1, z\}_{pbk(A)}}{\{z\}_{pbk(A)}}$;

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 above in which the roles and nonces are instantiated w.r.t. the principals of the session. Thus, the exponants of a nonce N_i^{QR} allow to distinguish nonces of different sessions. This distinction is not useful for the nonces of the intruder.

Additionally, the verification tool requires a constraint $\mathcal{C}(E)$ 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 $\{N_2, pvk(A)\}$.

3 Verification based on patterns keeping secrets

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.

3.1 Hat-messages: messages keeping secrets under a hat

If a principal A wants to protect a secret s, then he has to use a key whose inverse is not known by the intruder in order to encrypt every occurence of s in every message sent. The secret s itself need not be directly encrypted. Indeed, it should be enough that it 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, as honest principals following the protocol can unwillingly help the intruder in decrypting the message.

In order to develop this idea formally we need to introduce a few definitions. Recall that critical and non-critical positions as well as the notation $\in_c$ has been introduced in Section 2.2.

Definition 3.1 *We call* hat-term *any term of the form* $\mathbf{encr}(t,k)$. *It is called* hat-message *if it is a ground term. Then, a secret s is protected by a set of hat-messages H in a set of messages M, denoted by $M\langle H\rangle s$, if the following condition is satisfied:*

> *for all messages $m \in M$, for all critical positions p in m with $m_{|p} = s$, there exists a position $q \prec p$ such that $m_{|q} \in H$.* □

Example 3.1 According to our definition, the hat-message $\mathbf{encr}(s,k)$ protects the secret s in the messages $\mathbf{encr}(\mathbf{encr}(s,k),k')$ but it does not protect it neither in the message $\mathbf{encr}(\mathbf{encr}(s,k'),k)$ nor in $\mathbf{pair}(\mathbf{encr}(s,k),s)$. Furthermore, even if the key k is part of the secrets, it does not need to be protected in the previous messages for it never appears in critical position. □

In order to extend the previous definition to a set of secrets, we introduce a function $\mathcal{H} : \mathcal{T}(\mathcal{F}) \to \mathscr{P}(\mathcal{T}(\mathcal{F}))$ which associates each secret with a set of hat-messages. Since there will be no ambiguity, we overload the notation of Definition 3.1. Let $\mathcal{S}$ be a set of secrets, we define : $M\langle\mathcal{H}\rangle\mathcal{S} = \bigwedge_{s\in\mathcal{S}} M\langle\mathcal{H}(s)\rangle s$.

The notion of a message being protected by a hat-message introduced above does not take into account the capabilities of the intruder to decompose and compose new messages. 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, let us define the weak closure under decomposition of a term. Let T be a set terms. Then, T is called *weakly closed under decomposition*, or just weakly closed for short, if for any term $f(t_1, \cdots, t_n) \in T$ there is some $i \in \mathbb{N}_n$ such that $t_i \in T$. Then, a set T' is called a *weak closure* of T, if it is weakly closed, contains T and no proper sub-set of it satisfies these properties. Now, we are ready to extend the notion of a message being protected by a hat-message taking into account the intruder.

Definition 3.2 *A set $\mathcal{S}$ of messages is called* strongly protected *in M, denoted by $M[\mathcal{H}]\mathcal{S}$, if the following conditions are satisfied:*

(i) $M \vdash m \Rightarrow m\langle\mathcal{H}\rangle\mathcal{S}$, *i.e., the secrets in* $\mathcal{S}$ *must be protected in any message that the intruder can deduce from* M.

(ii) $\mathcal{S}$ *is weakly closed.*

(iii) $\bigcup_{s\in\mathcal{S}} Keys(\mathcal{H}(s))^{-1} \subseteq \mathcal{S}$ *where* $Keys(H)^{-1} = \{k^{-1} \mid \mathbf{encr}(t,k) \in H\}$, *i.e., the inverses of all the keys used in hat-messages are also secrets.*

□

Intuitively, condition *(ii)* ensures that the intruder will always miss at least one part of a secret preventing him from deducing it by composition. Condition *(iii)* ensures that the intruder will not be able to decrypt a secret protected by a hat-message.

We can relate the predicates $\vdash$, $\vdash^{\in_c}$ and $m[\mathcal{H}]\mathcal{S}$ a follows. Let the hat function $\widehat{\emptyset}$, be defined by the property $\forall t \in \mathcal{T}(\mathcal{F}).\ \widehat{\emptyset}(t) = \emptyset$. Then, we have:

Lemma 3.1 **a)** $M \vdash s \Leftrightarrow \neg\big(\exists \mathcal{H}, \mathcal{S}.\ s \in \mathcal{S} \wedge M[\mathcal{H}]\mathcal{S}\big)$ **b)** $M[\widehat{\emptyset}]s \Leftrightarrow M \not\vdash^{\in_c} s$

Our main motivation in introducing $M[\mathcal{H}]\mathcal{S}$ lays in the following Proposition that states that adding a message m in which the secrets of $\mathcal{S}$ are protected preserves strong protection of $\mathcal{S}$.

Proposition 3.1 *Let* $M, \mathcal{S} \subseteq \mathcal{T}(\mathcal{F})$ *be sets of messages,* m *be a message and* $\mathcal{H} : \mathcal{T}(\mathcal{F}) \to \mathscr{P}(\mathcal{T}(\mathcal{F}))$. *If* $M[\mathcal{H}]\mathcal{S}$ *and* $m\langle\mathcal{H}\rangle\mathcal{S}$ *then* $M \cup \{m\}[\mathcal{H}]\mathcal{S}$. □

The proof of this proposition is presented in the appendix. Notice while reading the proof that it does not rely on the fact that keys are atomic. Actually, our method can be extended to cover the case of non atomic keys.

Example 3.2 Consider a composed secret $s = \mathbf{pair}(s_1, s_2)$ and a $\mathcal{H}$ function defined by:

$$\mathcal{H}(s_1) = \mathcal{H}(s) = \{\mathbf{encr}(\mathbf{pair}(s_1, s_2), k)\}$$
$$\mathcal{H}(s_2) = \mathcal{H}(s_1) \cup \{\mathbf{encr}(s_2, k)\}$$
$$\mathcal{H}(k) = \mathcal{H}(k^{-1}) = \emptyset$$

Let $\mathcal{S} = \{k, k^{-1}, s, s_2\}$. Notice that $\mathcal{S}$ is weakly closed. Then, for the set of messages $M = \{\mathbf{encr}(\mathbf{pair}(s_1, s_2), k)\}$ and $m = \mathbf{pair}(s_1, \mathbf{encr}(s_2, k))$, we have $M[\mathcal{H}]\mathcal{S}$ and $m\langle\mathcal{H}\rangle s$. Indeed, k^{-1} does not appear at all; k is never used in critical positions; s and s_2 are protected in M by the hat $\mathbf{encr}(\mathbf{pair}(s_1, s_2), k)$; s_2 is protected in m by the hat $\mathbf{encr}(s_2, k)$; and s does not appear in m. Then, it follows from the proposition that $\mathcal{S}$ is protected by $\mathcal{H}$ in $M \cup \{m\}$, even if s_1 is not kept in m. Besides, notice that there is no hat term able to keep s_1 in m. □

Let now $r = t_1 \to t_2$ be a rule in $\mathcal{R}$ (recall that we have fixed an arbitrary protocol $P = (\mathcal{C}, \mathcal{R})$. We say that the pair $(\mathcal{S}, \mathcal{H})$ composed of a hat function and a set of secrets is *stable w.r.t. a rule* r, if for every subtitution σ, we have $\sigma(t_2)\langle\mathcal{H}\rangle\mathcal{S}$; it is stable w.r.t. a set of rules $\mathcal{R}$, if it stable w.r.t. to each rule in $\mathcal{R}$. Then, using Proposition 3.1, we can prove by induction the following Theorem:

Theorem 3.1 *Let* $\mathcal{S}$ *be a set of secrets,* $\mathcal{H}$ *be a hat function. If* $(\mathcal{S}, \mathcal{H})$ *is stable w.r.t. all rules in* $\mathcal{R}$ *and* $E_0[\mathcal{H}]\mathcal{S}$, *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 by* P.

3.2 Computing stable secrets and hat-messages

According to Theorem 3.1, given a protocol $P = (\mathcal{C}, \mathcal{R})$ and a set $\mathcal{S}$ of secrets, if we can find a hat function $\mathcal{H}$ and a set $\mathcal{S}'$ of secrets such that:

- the constraint $\mathcal{C}$ on E_0 – the messages initially known by the intruder – imply $E_0[\mathcal{H}]\mathcal{S}'$,

- $\mathcal{S} \subseteq \mathcal{S}'$ and
- $(\mathcal{S}', \mathcal{H})$ is stable w.r.t. $\mathcal{R}$,

we can conclude that the secrets in $\mathcal{S}$ are preserved by $P = (\mathcal{C}, \mathcal{R})$. In this section, we develop an algorithm that computes a stable pair $(\mathcal{S}', \mathcal{H})$. 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 hat-messages. Then, we define a symbolic representation for hat-messages and develop a symbolic algorithm.

3.3 A semantic version of the verification algorithm

In Figure 1, we present an algorithm that computes a pair $(\mathcal{S}, \mathcal{H})$ which is stable w.r.t. the rules of the protocol. It uses the function *Closure* that associates to a set T of terms a weak closure of T. The algorithm takes a set of rules $\mathcal{R}$, a set of secrets $\mathcal{S}$ and a function $\mathcal{H}$ as input. It is a fixpoint computation, starting with $(\mathcal{S}, \mathcal{H})$. If it terminates, it returns an augmented set of secrets $\mathcal{S}'$ and a function $\mathcal{H}'$ such that $\mathcal{H}'(s) \subseteq \mathcal{H}(s)$, for every secret s. We now explain the intuitively the clue point of the algorithm. Let us take a rule $t_p \to t_c$ in $\mathcal{R}$, a substitution $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F})$ and a hat-message h such that h protects a secret in $\sigma(t_p)$. If the secret is not protected by any hat-messages in $\sigma(t_c)$ – the conclusion of the instanciated rule, then the hat-message h is not safe indeed and it must be removed from the set of hat-messages. Actually, even when the inverse of the keys used in the hat-messages are unknown by the intruder, the secret can be unwillingly revealed by a principal. Think for instance of a protocol with $\{(y, x)\}_{pbk(A)} \to \{x\}_{pbk(y)}$ as a rule of principal A. On reception of the message $\{(I, Secret)\}_{pbk(A)}$, the principal A will respond $\{Secret\}_{pbk(I)}$ thus unwillingly decrypting the secret for the intruder. So, $\{(I, Secret)\}_{pbk(A)}$ is a particular case where $pbk(A)$ does not protect the secret and it must be removed from the set of hat-messages. Case 2 in the algorithm considers the case where a secret is not protected in the conclusion and the premisse does not contain a secret. In this case, the apparently harmless premise is as compromising as the secret, and hence, is added to the set of secrets.

The following proposition 3.2 summarizes the properties of the algorithm:

Proposition 3.2 *If the algorithm of Figure 1 applied to $(\mathcal{R}, \mathcal{S}, \mathcal{H})$ terminates, it returns $\mathcal{S}'$ and $\mathcal{H}'$ which satisfy the following conditions:*

1. *$(\mathcal{S}', \mathcal{H}')$ is stable w.r.t. $\mathcal{R}$,*
2. *$\mathcal{S} \subseteq \mathcal{S}'$ and*
3. *$Keys(\mathcal{H}')^{-1} \subseteq \mathcal{S}'$.* □

Using Proposition 3.2 and Theorem 3.1, we can prove the following:

Corollary 3.1 *If the algorithm of Figure 1 terminates with $(\mathcal{S}', \mathcal{H}')$ as result and each set of messages E_0 that satisfies $\mathcal{C}(E_0)$ also satisfies $E_0[\mathcal{H}']\mathcal{S}'$, we can conclude $\not\vdash_P \mathcal{S}'$, and hence, $\not\vdash_P \mathcal{S}$.* □

3.4 A symbolic representation of hat-messages: Pattern Terms

To develop an effective version of our semantic algorithm, we need to represent (potentially infinite) sets of hat-messages. To do so, we introduce a symbolic representation that consists in a pair $(\mathcal{G}, \mathcal{B})$ of sets of terms over variables in $\mathcal{X} \cup \mathcal{X}_s$. Here, $\mathcal{X}_s$ denotes a new disjoint set of variables. We use $x_s, y_s, \cdots$ as typical variables in $\mathcal{X}_s$. Roughly speaking, a hat-message for a secret is an instance of a term in $\mathcal{G}$ that is not an instance of any term in $\mathcal{B}$. Therefore, we call the terms in $\mathcal{G}$ *good patterns* and those in $\mathcal{B}$ *bad patterns*. In good patterns, we use the variables in $\mathcal{X}_s$ to mark the positions in which a term containing a secret is allowed to appear. While in bad patterns, they are used to mark positions where it is not guaranteed that the secret is not revealed. For instance, if $\mathcal{G}$ contains the term $\{x_s\}_K$ and $\mathcal{B}$

input: $\mathcal{R}$, a weakly closed $\mathcal{S}$ and $\mathcal{H}$
output: $\mathcal{S}'$, $\mathcal{H}'$ such that $(\mathcal{S}', \mathcal{H}')$ is stable w.r.t. $\mathcal{R}$.
$\mathcal{S}' := \mathcal{S}$; $\mathcal{H}' := \mathcal{H}$;
repeat
— *first, compute the closure that adds to S' one subpart of each compound secret of S'*
— *and add to the secrets the inverse of the keys used in hat-messages:*
$\mathcal{S}' := Closure(\mathcal{S}') \cup Keys(\mathcal{H}')^{-1}$; $\mathcal{S}_c := \mathcal{S}'$; $\mathcal{H}_c := \mathcal{H}'$;
for each $t_p \to t_c \in \mathcal{R}$
— *compute all Dangerous Substitutions of rule $t_p \to t_c$ where a secret is not kept in the conclusion*
$DS := \{\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F}) \mid \neg\big(\sigma(t_c)\langle\mathcal{H}'\rangle\mathcal{S}'\big)\}$;
— *compute the corresponding Dangerous Premisses*
$DP := \{\sigma(t_p) \mid \sigma \in DS\}$;
— *update the secret and hat-messages according to the dangerous premisses:*
— *Case 1) filter the hat-messages by removing those that appear in dangerous premisses*
for each $s \in \mathcal{S}'$, $m \in DP$ **do**
— *the Bad Hats-messages of s are those which protect a secret that can be disclosed by a rule*
$BHat := \{m_{|q} \mid \exists p.\ q \prec p \wedge m_{|p} = s \wedge m_{|q} \in \mathcal{H}'(s)\}$
— *select a subset of hat-messages to be removed*
choose $BHat' \subseteq BHat$ **with** $BHat \neq \emptyset \ \Rightarrow\ BHat' \neq \emptyset$
— *update the $\mathcal{H}$ function at point s*
$\mathcal{H}'(s) := \mathcal{H}'(s) \setminus BHat'$;
od
— *Case 2) add to the set of secrets all the bad premisses than do not contain any secret*
$newS := \{m \in DP \mid \forall s \in \mathcal{S}'.\ s \notin_c m\}$; $\mathcal{S}' := \mathcal{S}' \cup newS$
od
until $(\mathcal{S}', \mathcal{H}') = (\mathcal{S}_c, \mathcal{H}_c)$

Figure 1: The semantic version of the verification algorithm

contains the term $\{(A, x_s)\}_K$ then the message $\{(B, \mathit{Secret})\}_K$ will be in the set represented by $(\mathcal{G}, \mathcal{B})$, while the message $\{(A, \mathit{Secret})\}_K$ will not. Let us now define this symbolic representation formally.

To do so, we introduce *pattern terms* defined by the following BNF:

$$pt \quad ::= \quad N \mid P \mid K \mid x \mid x_s \mid \mathbf{pair}(pt_1, pt_2) \mid \mathbf{encr}(pt, K) \mid \mathit{Sup}\,(pt)$$

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

Intuitively, as can be seen from the following definition, $\mathit{Sup}\,(t)$ 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 $[\![\mathit{Sup}\,(A)]\!]$.

Definition 3.3 *Given a pattern term* pt, *let* $[\![\mathrm{pt}]\!]$ *be defined as follows:*

$$\begin{array}{rcl} [\![\mathrm{pt}]\!] & = & \{\mathrm{pt}\}\ ;\ \ \textit{if } \mathrm{pt} \textit{ is a constant or a variable} \\ [\![\mathbf{pair}(\mathrm{pt}_1, \mathrm{pt}_2)]\!] & = & \{\mathbf{pair}(t_1, t_2) \mid t_1 \in [\![\mathrm{pt}_1]\!], t_2 \in [\![\mathrm{pt}_2]\!]\} \\ [\![\mathbf{encr}(\mathrm{pt}_1, k)]\!] & = & \{\mathbf{encr}(t_1, k) \mid t_1 \in [\![\mathrm{pt}_1]\!]\} \\ [\![\mathit{Sup}\,(\mathrm{pt})]\!] & = & \{t \mid \ \textit{there is a position } p \textit{ in } t \textit{ s.t. } t_{|p} \in [\![\mathrm{pt}]\!]\} \end{array}$$

□

To define the concretization of pattern terms we need to introduce *positive* and *negative substitutions*. As usual a substitution is a mapping $\sigma : \mathcal{X} \cup \mathcal{X}_s \to \mathcal{T}(\mathcal{X} \cup \mathcal{X}_s, \mathcal{F})$. A ground substitution is a mapping $\sigma : \mathcal{X} \cup \mathcal{X}_s \to \mathcal{T}(\mathcal{F})$. A substitution is called *positive*, if $\sigma(x)_{|p} \notin \mathcal{X}_s \cup \mathcal{S}$, for every $x \in dom(\sigma)$ and critical position p in $\sigma(x)$. It is *negative*, if there is a variable $x_s \in \mathcal{X}_s \cap dom(\sigma)$ such that $\sigma(x_s)_{|p} \in \mathcal{X}_s \cup \mathcal{S}$, for some critical position p in $\sigma(x_s)$. Then, let $\gamma^+_{\mathcal{S}}(pt) = \{\sigma(t) \mid t \in [\![pt]\!], \sigma \text{ is ground and positive}\}$ and $\gamma^-_{\mathcal{S}}(pt) = \{\sigma(t) \mid t \in [\![pt]\!], \sigma \text{ is ground and negative}\}$. As usual, we can extend $\gamma^+_{\mathcal{S}}$ and $\gamma^-_{\mathcal{S}}$ pointwisely to sets of pattern terms. We use $\gamma^-_{\mathcal{S}}$ for the concretization of bad patterns and $\gamma^+_{\mathcal{S}}$ for the concretization of good patterns. We represent pairs $(\mathcal{S}, \mathcal{H})$ by triples $\langle \mathcal{S}, \mathcal{G}, \mathcal{B} \rangle$ of finite sets of secrets, good and bad patterns. Good patterns correspond to "maybe safe" hat terms and bad patterns define "really unsafe" hat terms. The concretization of $\langle \mathcal{S}, \mathcal{G}, \mathcal{B} \rangle$ is the pair $(\mathcal{S}, \mathcal{H})$, where $\mathcal{H}$ associates to every $s \in \mathcal{S}$ the same set $\gamma^+_{\mathcal{S}}(\mathcal{G}) \setminus \gamma^-_{\mathcal{S}}(\mathcal{B})$ of hat-messages. More formally, we have the following:

Definition 3.4 *A* symbolic representation *SR is triple $\langle \mathcal{S}, \mathcal{G}, \mathcal{B} \rangle$, where*

- *$\mathcal{S}$ is a finite set of terms that represents the secrets,*
- *$\mathcal{G}$ is a finite set of terms that represent the good hat messages, and*
- *$\mathcal{B}$ is a finite set of pattern terms that represent the bad hat messages*

A symbolic representation $\langle \mathcal{S}, \mathcal{G}, \mathcal{B} \rangle$ defines the pair $(\mathcal{S}, \mathcal{H})$ where $\mathcal{H}(s) = \gamma^+_{\mathcal{S}}(\mathcal{G}) \setminus \gamma^-_{\mathcal{S}}(\mathcal{B})$, for each $s \in \mathcal{S}$.
□

Example 3.3 Let $g = \mathbf{encr}(x_s, K)$ and $SR = \langle \{N, K^{-1}\}, \{g\}, \emptyset \rangle$. Then, the concretization of SR is $(\{N, K^{-1}\}, [N \mapsto H, K^{-1} \mapsto H])$, where H consists of all ground instances of g that contains N or K^{-1}. Thus, N is protected in $\mathbf{encr}(\ \mathbf{encr}(\mathbf{encr}(N, K''), K),\ K')$, for the hat message $\mathbf{encr}(\mathbf{encr}(N, K''), K)$ belongs to $\mathcal{H}(N) = H$ as an instance of g with the substitution $[x_s \mapsto \mathbf{encr}(N, K'')]$. □

4 A symbolic verification algorithm

The symbolic algorithm is obtained from the algorithm of Figure 1 by replacing each operation by a corresponding symbolic one that operates on $\langle \mathcal{S}, \mathcal{G}, \mathcal{B} \rangle$. For the sake of presentation, we first explain the symbolic algorithm in the particular case where $\mathcal{B}$ consists of terms rather than pattern terms, i.e., *Sup* does not occur in any term in $\mathcal{B}$. We will explain later how it extends to pattern terms and what are the difficulties to solve.

4.1 The algorithm on terms

Before presenting the algorithm we need to introduce the following definitions. Let t and t' be terms. A substitution $\sigma : \mathcal{X} \cup \mathcal{X}_s \to \mathcal{T}(\mathcal{X} \cup \mathcal{X}_s, \mathcal{F})$ is called *a positive matcher* of t' on t, if it is a positive substitution with $\sigma(t') = t$. Given a set $\mathcal{G}$ of terms, we say that t matches positively with $\mathcal{G}$, if there is a term $t' \in \mathcal{G}$ and a positive matcher of t' on t. Moreover, a substitution $\sigma : \mathcal{X} \cup \mathcal{X}_s \to \mathcal{T}(\mathcal{X} \cup \mathcal{X}_s, \mathcal{F})$ is called *a negative unifier* of t' and t, if it is a negative substitution with $\sigma(t') = \sigma(t)$.

The symbolic algorithm takes as input a set of rules $\mathcal{R}$, a set of secrets $\mathcal{S}$, a set of good terms $\mathcal{G}$ and an empty set of bad terms $\mathcal{B} = \emptyset$. It computes new bad terms and secrets until the concretization of $\langle \mathcal{S}, \mathcal{G}, \mathcal{B} \rangle$ becomes stable w.r.t. all rules in $\mathcal{R}$. Let us now sketch its main steps:

1. The set $\mathcal{S}$ of secrets becomes $\mathcal{S} \cup Keys(\mathcal{G})^{-1}$, where $Keys(\mathcal{G})^{-1}$ is the set of keys of the form k^{-1} such that $\mathbf{encr}(t, k)$ is in $\mathcal{G}$.
2. Given a rule $t_p \to t_c$ in $\mathcal{R}$, we have to consider all possible occurrences of a secret in the conclusion t_c. This is done by replacing variables in t_c by secret variables. In other words, we consider all rules in $\mathcal{R}' = \{\sigma(t_p) \to \sigma(t_c) \mid t_p \to t_c \in \mathcal{R},\ \sigma : var(t_c) \to \mathcal{X}_s\}$.

3. Given a rule $t_p \to t_c$ in $\mathcal{R}'$, the algorithm computes the finite set of dangerous substitution DS as follows. A substitution $\sigma : \mathcal{X} \cup \mathcal{X}_s \to \mathcal{T}(\mathcal{X} \cup \mathcal{X}_s, \mathcal{F})$ is *dangerous* if there is a critical position p in t_c such that $t_{c|p} \in \mathcal{X}_s \cup \mathcal{S}$ and for every position $q \prec p$, when $t_{c|q}$ matches positively with $\mathcal{G}$, it also unifies by σ with a term in $\mathcal{B}$. Then, $DS := \{\sigma : \mathcal{X} \cup \mathcal{X}_s \to \mathcal{T}(\mathcal{X} \cup \mathcal{X}_s, \mathcal{F}) \mid \sigma \textit{ is dangerous}\}$. The paragraph below illustrates the computation of dangerous substitutions.

4. The set of *bad premises* is given by $BP = \{\sigma(t_p) \mid \sigma \in DS\}$. The new bad terms are the subterms of a bad premisse $t \in BP$ that were supposed to protect a secret in t as they match positively with $\mathcal{G}$. So, bad terms are particular instances of terms in $\mathcal{G}$ that must be removed. Formally,

$$BHat = \{t_{|q} \mid t \in BP,\ \exists p.\ q \prec p\ \wedge\ t_{|p} \in \mathcal{X}_s \cup \mathcal{S}\ \wedge\ t_{|q} \textit{ matches positively with } \mathcal{G}\}$$

   Notice that we use the same set of hat-terms for all the secrets, so each occurrence of $\mathcal{H}'$ and $\mathcal{H}'(s)$ is now replaced by the set of bad terms $\mathcal{B}'$ and the restriction of $\mathcal{H}'(s)$ is replaced by: $\mathcal{B}' := \mathcal{B}' \cup BHat$.

5. Finally, $newS$ is replaced by $newS := \{t \in DP \mid var(t) \cap \mathcal{X}_s = \emptyset,\ \forall s \in \mathcal{S}'.\ s \notin_c t\}$.

Computation of dangerous substitutions

We present the algorithm that computes the dangerous substitutions induced by a rule $t_p \to t_c$, a set of good terms $\mathcal{G}$ and a set of bad terms $\mathcal{B}$. We apply the following algorithm to each secret (a message in $\mathcal{S}$ or a variable in $\mathcal{X}_s$) that appears at critical position in the conclusion of the rule.

Let p be a position of a secret in t_c. Let $\mathcal{PP}$ be the set of positions p_i above p such that a good term of $\mathcal{G}$ matches positively with $t_{c|p_i}$. In the sequel, the found positive matchers are of no use; we only retain the protecting positions $p_1 \prec p_2 \prec \ldots \prec p_n (\prec p)$. We define below the function Φ that computes all the negative unifiers that cancel each protecting position by a bad term. Formally, the dangerous substitutions are the negative unifiers σ that satisfy: $\bigwedge_{p_i \in \mathcal{PP}} \sigma(t_c)_{|p_i} = \sigma(b_i)$ with $b_i \in \mathcal{B}$.

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 bad term.

$$\Phi(t_c,\ \mathcal{B},\ \mathcal{PP},\ DS) = \begin{cases} \{\sigma \in DS \mid \sigma \textit{ is a negative unifier}\} & \text{if } \mathcal{PP} = \emptyset \\ \Phi(t_c,\ \mathcal{B},\ \mathcal{PP} \setminus \{p_i\},\ \bigcup\limits_{\sigma_j \in DS} \{\sigma_j \cup \sigma_1^{i,j}, \cdots, \sigma_j \cup \sigma_{n_{i,j}}^{i,j}\}\ ) & \text{for a position } p_i \text{ chosen in } \mathcal{PP} \end{cases}$$

where the $\sigma_k^{i,j}$ in the fourth argument are the negative unifiers resulting of the unification of $\sigma_j(t_c)_{|p_i}$ with some bad terms of $\mathcal{B}$.

The same algorithm is used in the case where $\mathcal{G}$ and $\mathcal{B}$ are sets of terms or sets of patterns terms. We only need to adapt the unification algorithm. In the case of terms, we use the standard *most general unifier*; and for patterns terms, we define a unification algorithm presented in Section 4.2.

Example 4.1 We illustrates the computation of dangerous substitutions on the sets of terms $\mathcal{G} = \{\ \mathbf{encr}(x_s, K_B),\ \mathbf{encr}(x_s, K_A)\ \}$; $\mathcal{B} = \{\ \mathbf{encr}(\mathbf{pair}(I, x_s'), K_B),\ \mathbf{encr}(\mathbf{pair}(\mathbf{pair}(A, x''), x_s''), K_B)\ \}$; and a rule $t_p \to t_c$ given in Figure 2.

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 secret variable appears. We find one variable $y_s \in \mathcal{X}_s$ at position 0.1.1.0.1 in the term t_c. For each critical position, we look for the positions above it that match with a good term in $\mathcal{G}$. For y_s, we found exaclty two protecting positions: the term $\mathbf{encr}(x_s, K_B)$ matches positively at position $p_1 = \epsilon$ and also at position $p_2 = 0.1.1$. Then, the function Φ looks for all substitutions that negatively unify some bad terms with the terms at the protecting positions p_1 and p_2. Starting with position $p_1 = \epsilon$, it unifies $t_{c|\epsilon}$ with the bad term $\mathbf{encr}(\mathbf{pair}(I, x_s'), K_B)$ for the negative unifier $\sigma' = [z = I, x_s' = t_{c|0.1}]$. This cancels the top most protection. Then, the function Φ attempts

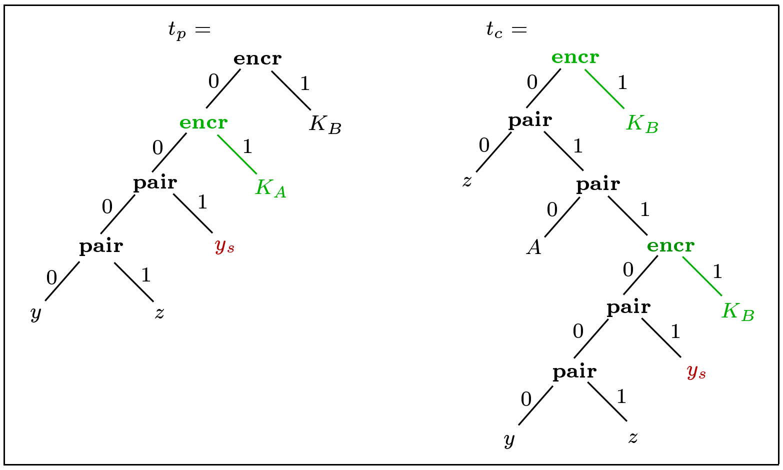

Figure 2: Illustration of computing dangerous substitutions.

to complete the substitution σ' so that it also cancels the protection at position $p_2 = 0.1.1$. To do so, it tries to unify the term $\sigma'(t_c)_{|p_2} = \mathbf{encr}(\mathbf{pair}(\mathbf{pair}(y, I), y_s), K_B)$ with a bad term and succeeds with the bad term $\mathbf{encr}(\mathbf{pair}(\mathbf{pair}(A, x''), x''_s), K_B)$ and the negative unifier $\sigma'' = [y = A, x'' = I, y_s = x''_s]$. The two unifiers are then composed and restricted to the domain $var(t_c)$ resulting in 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 the set of dangerous substitutions $\{\sigma\}$. We now look at the premise of the rule to compute the new bad terms induced by σ. The position of the secret variable y_s in t_p is protected by the good term $\mathbf{encr}(x_s, K_A)$ that matches positively on $t_{p|0}$. However, the dangerous substitution σ tells that this protection will not work in case where y is A and z is I. Consequently, we refine the set of hat messages by removing this particular case. In our symbolic representation, this comes out to add $\sigma(t_{p|0}) = \mathbf{encr}(\ \mathbf{pair}(\mathbf{pair}(A, I), y_s),\ K_A)$ to the set of bad terms. □

4.2 Dealing with pattern terms

First of all, it is worth to mention that pattern terms are more expressive than terms, that is, there are sets of messages that can be described as pattern terms but not as terms. This is for instance the case for the set of messages that contain the constant A as 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 pattern terms to represent the bad hat-messages.

Unification and matching are the key operation in the 3^{rd} anf 4^{th} steps of the symbolic algorithm. The problem we need to solve for obtaining our symbolic algorithm is, however, *not* unification of pattern terms. The problem we need to solve is the following: Given two pattern terms u and t, 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 pattern terms u and t is simply a term, that is, without occurence of *Sup* in it [1]. 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 1 for the notations). The set of pattern terms induced by $\mathcal{F}$ and $\mathcal{X}$, denoted by $\mathcal{PT}(\mathcal{F}, \mathcal{X})$ is defined by the following BNF:

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

[1] Indeed, we need to unify the conslusion of a rule, which is a term, with a bad pattern.

where x is a variable in $\mathcal{X}$ and 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 pattern term t is a set of terms in $\mathcal{T}(\mathcal{X}, \mathcal{F})$ defined in the same way as in Definition 3.3.

Definition 4.1 *Given two pattern terms u and t, a substitution $\sigma : \mathcal{X} \to \mathcal{PT}(\mathcal{X}, \mathcal{F})$ is a called a* maximal general unifier *for u and t, if the following conditions are satisfied:*

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

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

In general there will be more than one maximal general unifier for u and t even modulo renaming. The definition of $\mathcal{U}$ can be extended in the usual way –as for unification– to sets $\{(u_i, t_i) \mid i \in [1, n]\}$ of pairs of pattern terms. In the sequel, we prefer to write $u_i = t_i$ instead of (u_i, t_i) as our algorithm essentially consists in manipulating some kind of equations.

In this section, we want to develop an algorithm that given $E = \{u_i = t_i \mid i \in [1, n]\}$ 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. [3]) will give the solution.

An equation $u = t$ is called simple, if the argument of any occurence of *Sup* in u and t is either a constant or a variable. A gep E is called simple, if all its equations are simple. It is not difficult to see that every gep E can be transformed into an equivalent simple gep E'. Equivalent here means that $\mathcal{U}(E')$ restricted to the variables in E is the same as $\mathcal{U}(E)$.

Example 4.2 *Consider the following gep $E = \{Sup\,(f(b, Sup\,(g(x)))) = f(b, g(Sup\,(a)))\}$. Then, E is equivalent to E' that consists of the following equations:*

$$Sup\,(x_0) = f(b, g(Sup\,(a))),\ x_0 = f(b, Sup\,(x_1)),\ x_1 = g(x).$$

□

We first recall in Figure 3 the usual rules for solving unification.

Delete	$\{t = t\} \cup E$	$\Rightarrow\ E$
Decompe	$\{f(u_1, \cdots, u_n) = f(t_1, \cdots, t_n)\} \cup E$	$\Rightarrow\ \{u_i = t_i \mid i \in [0, n]\} \cup E$
Orient	$\{t = x\} \cup E$	$\Rightarrow\ \{x = t\} \cup E$
Eliminate	$\{x = t\} \cup E$	$\Rightarrow\ \{x = t\} \cup E[t/x]$
Clash	$\{f(u_1, \cdots, u_n) = g(t_1, \cdots, t_m)\} \cup E$	$\Rightarrow\ \mathcal{U} = \emptyset$
Occurs-Check	$\{x = t\} \cup E$	$\Rightarrow\ \mathcal{U} = \emptyset$, if x is not t and $x \in var(t)$

Figure 3: Rules of solving unification

Now to deal with *Sup*, we add new rules. We attract the reader's attention to the fact that the first rule (Splitting) transforms a simple gep E into a set of simple 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. The rules for *Sup* are presented in Figure 4.

Example 4.3 *Reconsidering Example 4.2, our algorithm yields $\{x = Sup\,(a)\}$. It can be easily verified that indeed $\mathcal{U}(E) = \{x = Sup\,(a)\}$.*

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

Splitting-var	$\{Sup\,(x) = f(t_1, \cdots, t_n)\} \cup E$	$\Rightarrow$	$\{x = t'\} \cup E$, for $t' \leq f(t_1, \cdots, t_n)$
Splitting-const	$\{Sup\,(a) = f(t_1, \cdots, t_n)\} \cup E$	$\Rightarrow$	$\{a = t'\} \cup E$, for $t' < f(t_1, \cdots, t_n)$
Simplification	$\{t' = Sup\,(t), t' = t\} \cup E$	$\Rightarrow$	$\{t' = t\} \cup E$
Sup-Sup-var-const-2	$\{Sup\,(x) = Sup\,(a)\} \cup E$	$\Rightarrow$	E, if $\bigcup_{i\geq 2} \mathcal{F}^{(i)} \neq \emptyset$
Sup-Sup-var-const-1	$\{Sup\,(x) = Sup\,(a)\} \cup E$	$\Rightarrow$	$\{x = Sup\,(a)\} \cup E$, if $\bigcup_{i\geq 2} \mathcal{F}^{(i)} = \emptyset$
Sup-Sup-const-2	$\{Sup\,(a) = Sup\,(b)\} \cup E$	$\Rightarrow$	E, if $\bigcup_{i\geq 2} \mathcal{F}^{(i)} \neq \emptyset$
Sup-Sup-const-1	$\{Sup\,(a) = Sup\,(b)\} \cup E$	$\Rightarrow$	$\{a = b\} \cup E$, if $\bigcup_{i\geq 2} \mathcal{F}^{(i)} = \emptyset$
Sup-Sup-var-2	$\{Sup\,(x) = Sup\,(y)\} \cup E$	$\Rightarrow$	E, if $\bigcup_{i\geq 2} \mathcal{F}^{(i)} \neq \emptyset$
Sup-Sup-var-1	$\{Sup\,(x) = Sup\,(y)\} \cup E$	$\Rightarrow$	$\{x = Sup\,(y)\} \cup E$, $\{Sup\,(x) = y\} \cup E$, if $\bigcup_{i\geq 2} \mathcal{F}^{(i)} = \emptyset$

Figure 4: Rules for *Sup*

- m_1 is the number of variables for which there is no equation of the form $x = t$,
- m_2 is the size of E, i.e. $\Sigma_{u=t\in E}(|u| + |t|)$ and
- m_3 is the number of equations $t = x$ in E.

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

4.3 On the termination of the symbolic algorithm

The symbolic verification algorithm does not always terminate, even if it terminates in many examples of protocols. In this section, we present a technique that makes a depth-first implementation of the algorithm terminate more often. In fact, it terminates on all pratical examples of protocols we tried on HERMES, a prototype implementation of our verification algorithm (see Table 1).

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

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

Let us consider an example to clarify these definitions.

Example 4.4 *Consider the following rule from the session (A, A) of Needham-Schroeder-Lowe protocol presented in Section 4.4:*

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

Consider the sequence $(\{\theta^i(I, x)\}_{K_A})_{i\geq 0}$, where $\theta(z) = (A, (N_1^{AA}, z))$. The first three terms of the sequence are $\{\theta^0(I, x)\}_{K_A} = \{(I, x)\}_{K_A}$, $\{\theta^1(I, x)\}_{K_A} = \{(A, (N_1^{AA}, (I, x)))\}_{K_A}$ and $\{\theta^2(I, x)\}_{K_A} = \{(A, (N_1^{AA}, (A, (N_1^{AA}, (I, x)))))\}_{K_A}$. The whole sequence can be obtained by iteratively computing the bad patterns induced by the rule r starting from the bad term $\{(I, x)\}_{K_A}$. Thus, a naive application of our symbolic algorithm will not terminate. On the other hand, this sequence is increasing at $(p_i = 0(11)^i)_{i\geq 0}$. Indeed, $\{\theta^i(I, x)\}_{K_A}[z/p_0] = \{z\}_{K_A}$ and $\left(\{\theta^i(I, x)\}_{K_A}\right)_{|p_i} = (I, x)$, for every $i \geq 0$. We will see now how this fact can be exploited to make the algorithm to converge. □

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

Proposition 4.1 *Let $(t_i)_{i\geq 0}$ be increasing at $(p_i)_{i\geq 0}$. Then,*

$$\bigcup_{i\geq 0}[\![t_i]\!] \subseteq \bigcup_{i<j}[\![t_i]\!] \cup [\![t_j[Sup\,(t_{j|p_j})/p_j]]\!], \text{ for every } j \geq 0.$$

□

Example 4.5 *Consider again our Example 4.4. Then, if we choose $j = 1$, we obtain a set consisting of the two pattern terms $\{(I, x)\}_{K_A}$ and $\{(A, (N_1^{AA}, Sup\,(I, x)\,))\}_{K_A}$ which approximates the whole sequence $\left(\{\theta^i(I, x)\}_{K_A}\right)_{i\geq 0}$.*

Protocol Name	Result	Time (s)
Yahalom	OK	13.81
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
Kao-Chow	Attack	0.93
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	0.02

Table 1: The results provided by HERMES, our prototype for verifying secrecy properties

4.4 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:

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

We run our verification algorithm with $\mathcal{S} = \{N_2, K_A^{-1}, K_B^{-1}\}$, the empty set of bad patterns and the set $\mathcal{G} = \{\{x_s\}_{K_A}, \{x_s\}_{K_B}\}$ of good patterns. The algorithm terminates with the set of secrets unchanged and the set $\mathcal{B}$ of bad patterns given in Figure 5. As the initial constraints are $E \not\vdash^{\in_c} \{N_1, N_2, K_A^{-1}\}$,

$$\boxed{\{I, x_s\}_{K_A}, \quad \{A, (N_1^{AA}, Sup\,(I, x_s))\}_{K_A}, \quad \{A, (N_1, Sup\,(I, x_s))\}_{K_A}.}$$

Figure 5: The bad pattern-terms for the Needham-Schreoder-Lowe protocol

that is, none of the messages in $\{N_1, N_2, K_A^{-1}\}$ is contained at a critical position in a message derivable

[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.

from E, it is easy to prove that we have $E[\gamma_S^+(\mathcal{G}) \setminus \gamma_S^-(\mathcal{B})]\, \mathcal{S}$. Hence, we can conclude that the Needham-Schroeder-Lowe protocol preserves the secret N_2. Concerning, the uncorrected version of Example 2.1, during computation of new secrets and bad 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 construct an attack. This way, we obtain the attack known as "man in the middle". $\square$

5 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 [2, 26, 20], 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 [1]. The main contribution of the paper is a verification algorithm that consists of computing an inductive invariant using patterns 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.

A interactive version of our tool together with the examples of Table 1 is available at the url `http://www-verimag.imag.fr/~lbozga/hermes/hermes.php`.

References

[1] Roberto M. Amadio and Witold Charatonik. On name generation and set-based analysis in dolev-yao model. Technical Report 4379, INRIA, 2002.

[2] Roberto M. Amadio and Denis Lugiez. On the reachability problem in cryptographic protocols. In *International Conference on Concurrency Theory*, volume 1877, pages 380–394, 2000.

[3] Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

[4] D. Bolignano. Integrating proof-based and model-checking techniques for the formal verification of cryptographic protocols. *Lecture Notes in Computer Science*, 1427:77–??, 1998.

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

[6] L. Bozga, Y. Lakhnech, and M. Périn. Abstract interpretation for scerecy using patterns. Technical report, Verimag, 2002.

[7] J. Clark and J. Joacob. A survey on authentification protocol. http://www.cs.york.ac.uk/j̃ac/papers/drareviewps.ps, 1997.

[8] 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 (PROCOMET)*, 1998.

[9] H. Comon, V. Cortier, and J. Mitchell. Tree automata with one memory, set constraints, and ping-pong protocols. In *Automata, Languages and Programming, 28th International Colloquium, ICALP 2001, Crete, Greece, July 8-12, 2001, Proceedings*, volume 2076 of *Lecture Notes in Computer Science*. Springer, 2001.

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

[11] V. Cortier, J. Millen, and H. Rueß. Proving secrecy is easy enough. In *Proc. 14th IEEE Computer Security Foundations Workshop (CSFW'01), Cape Breton, Nova Scotia, Canada, June 2001*, pages 97–110. IEEE Comp. Soc. Press, 2001.

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

[13] N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. Undecidability of bounded security protocols. In N. Heintze and Proceedings E. Clarke, editors, editors, *Workshop on Formal Methods and Security Protocols — FMSP, Trento, Italy, July 1999.*, 1999.

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

[15] T. Genet and F. Klay. Rewriting for Cryptographic Protocol Verification. In *Proceedings 17th International Conference on Automated Deduction*, volume 1831 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 2000.

[16] Jean Goubault-Larrecq. A method for automatic cryptographic protocol verification. In Dominique Méry Beverly Sanders, editor, *Beverly Sanders , Dominique M Fifth International Workshop on Formal Methods for Parallel Programming: Theory and Applications (FMPPTA 2000)*, number 1800 in Lecture Notes in Computer Science. Springer-Verlag, 2000.

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

[18] G. Lowe. Breaking and fixing the needham-schroeder public-key protocol using fdr. In *TACAS*, number 1055 in Lecture Notes in Computer Science, pages 147–166, 1996.

[19] C. Meadows. Invariant generation techniques in cryptographic protocol analysis. In *PCSFW: Proceedings of The 13th Computer Security Foundations Workshop*. IEEE Computer Society Press, 2000.

[20] J. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *roc. 8th ACM Conference on Computer and Communications Security (CCS '01)*, pages 166–175, 2001.

[21] J. C. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using mur. In *Proceedings of the 1997 Conference on Security and Privacy (S&P-97)*, pages 141–153, Los Alamitos, May 4–7 1997. IEEE Press.

[22] David Monniaux. Decision procedures for the analysis of cryptographic protocols by logics of belief. In *12th Computer Security Foundations Workshop*. IEEE, 1999.

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

[24] L. Paulson. Proving properties of security protocols by induction. In *10th IEEE Computer Security Foundations Workshop (CSFW '97)*, pages 70–83, Washington - Brussels - Tokyo, June 1997. IEEE.

[25] D. Prawitz. *Natural Deduction: A Proof-Theoretical Study*. Almquivst & Wikseel, 1965.

[26] M. Rusinowitch and M. Turuani. Protocol insecurity with finite number of sessions is np-complete. In *14th IEEE Computer Security Foundations Workshop (2001), pp. 174–190.*, 2001.

[27] S. Schneider. Verifying authentication protocols with CSP. In *10th IEEE Computer Security Foundations Workshop (CSFW '97)*, pages 3–17, Washington - Brussels - Tokyo, June 1997. IEEE.

A Proof of the main proposition

Let us first recall the proposition.

Proposition A.1 *Let $M, S \subseteq \mathcal{T}(\mathcal{F})$ be sets of messages, m be a message and $\mathcal{H} : \mathcal{T}(\mathcal{F}) \to \mathscr{P}(\mathcal{T}(\mathcal{F}))$. If $M[\mathcal{H}]S$ and $m\langle\mathcal{H}\rangle S$ then $M \cup \{m\}[\mathcal{H}]S$.*

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

Assume that $M \cup \{m\}[\mathcal{H}]S$ does not hold. Then, there exists a derivation-minimal counter-example m'. The existence of m' can be proved as follows. Take a derivation of $M, m \vdash m'$ and let N_0 be its size. If m' is not a derivation-minimal counter-example then there must exist a sub-derivation $M, m \vdash m'_1$ with $\neg m'_1\langle\mathcal{H}\rangle S$. Clearly, the size N_1 of the derivation tree of m'_1 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 $\mathbb{N}$.
Thus, let us come back to our derivation-minimal counter-example m'. We derive a contradiction by case analysis on the last derivation step in $M, m \vdash m'$.

1. $m' \in M \cup \{m\}$. This, contradicts the assumptions $M[\mathcal{H}]S$ and $m\langle\mathcal{H}\rangle S$.
2. Case of encryption: Thus, $m' = \{s\}_k$, $M \cup \{m\} \vdash s$ and $M \cup \{m\} \vdash k$. Since m' is a derivation-minimal counter-example, we have $m' \in S$. Hence, $s \in S$ or $k \in S$ which contradicts the derivation-minimality of m'.
3. Case of pairing: Similar to the previous case.
4. Case of projection: This also contradicts the derivation-minimality assumption.
5. Case of decryption: Then, $M \cup \{m\} \vdash \{m'\}_{k'}$ and $M \cup \{m\} \vdash k'^{-1}$. If $k'^{-1} \notin S$, then $\neg\{m'\}_{k'}\langle\mathcal{H}\rangle s'$ which contradicts the derivation-minimality of m'. If $k'^{-1} \in S$ then $M \cup \{m\} \vdash k'^{-1}$ contradicts the derivation-minimality of m'.

$\square$