Decidable Analysis for a Class of Cryptographic Group Protocols with Unbounded Lists

Najah Chridi
LORIA-UHP
Nancy, France
Email: chridi@loria.fr

Mathieu Turuani
LORIA-INRIA
Nancy, France
Email: turuani@loria.fr

Michael Rusinowitch
LORIA-INRIA
Nancy, France
Email: rusi@loria.fr

Abstract

Cryptographic protocols are crucial for securing electronic transactions. The confidence in these protocols can be increased by the formal analysis of their security properties. Although many works have been dedicated to standard protocols like Needham-Schroeder very few address the more challenging class of group protocols. We have introduced in previous work [6] a synchronous model for group protocols, that generalizes standard protocol models by permitting unbounded lists inside messages. This approach also applies to analyzing Web Services manipulating sequences of items. In this model we propose now a decision procedure for the sub-class of Well-Tagged protocols with Autonomous Keys.

Introduction

Cryptographic protocols are crucial for securing electronic transactions. They rely on cryptographic functions to ensure security properties such as secrecy or authentication. The confidence in these protocols can be increased by a formal analysis in order to verify that the security properties are met at least at the logical level, that is, even when abstracting from the cryptographic functions and considering messages as first-order terms. Verification at the logical level is nevertheless a non-trivial task since cryptographic protocols are infinite state systems and, for instance, the set of potential messages that can be generated by an intruder is unbounded. Recently numerous works have been dedicated to the design of automated verification tools for cryptographic protocols. Such tools are often based on model-checking, modal logics, equational reasoning, and resolution theorem-proving (see e.g., [14], [27], [1]). Checking whether a protocol is flawed in the abstract Dolev Yao model ([8]) can often be reduced to a constraint solving problem in a term algebra (modulo an equational theory). This constraint-based approach has proved to be quite effective on standard benchmarks and has also permitted the discovery of new flaws in several protocols (see e.g., [3]).

This work was supported by AVANTSSAR, FP7-ICT-2007-1 Project No.216471.

However to our knowledge it has never been applied to the more challenging group protocols [18]. In fact very few formal verification results are available for such protocols. Indeed, these protocols and more specifically protocols for group key agreement often work by combining contributions from each member of the group into the group key. Modelling such protocols generally requires unbounded lists. The difficulty of the verification of these protocols relies then to the fact that they may perform an arbitrary number of steps since the group of communicating agents is a priori unbounded. They may also involve iterative or recursive computations. All these features allow one to encode easily undecidable problems.

Related work. Several works have considered protocols with unbounded number of participants and recursive steps. The formal analysis of such protocols goes back to [19] who studied the Recursive Authentication (RA) protocol of [9] for an unbounded number of participants using Isabelle/HOL theorem prover. However if the protocol is deficient there is no automatic mechanism to find the attack.

The validation of group protocols has been investigated in the CLIQUES project ([24]), based on group Diffie-Hellman (A-GDH) protocols. Several analysis methods have been applied in this project, from manual to automatic ones. An interesting result in this area has been obtained by [20] who found several attacks on the CLIQUES suite. Recently, [11] have developed an automata-based approximation technique to analyze this class of protocols and check the absence of flaw in presence of a **passive intruder**. The work [15] provides a formal specification of GDOI, a group protocol being proposed as an IETF standard, to be used with the NRL Protocol Analyzer [16]. [23] have used Coral system to analyze an improved version of the multicast group key management protocol by Tanaka and Sato ([25]). Two serious attacks have been found on this protocol. Coral has also discovered other attacks on Asokan-Ginzboorg protocol([2]) and Iolus protocol([17]). Other attacks were found in [7].

Some works have focused on the modelling of recursive computations performed by some participants (such as a server) in group protocols. [13] introduce tree transducers

to model recursion and to allow the protocol participants to output structured messages. This work gives a decision algorithm for secrecy in the case of atomic keys and bounded message size in the Dolev Yao setting. However messages cannot be tested for equality without losing decidability. Similarly using composed keys or adding equational theories for XOR or Diffie-Hellman exponentiation in their model leads to undecidability. [26] introduces a class of Horn clauses to model the recursive behavior of participants. In this model protocol participants may receive messages of unbounded sizes, send multiple messages in a single step, compare and store messages. He gives a decision procedure to check whether protocols in this model satisfy secrecy properties. His algorithm is in NEXPTIME and is based on the derivation of an exponential bound on the size of minimal attacks. Hence this nice result is rather of theoretical flavor and is not suitable for an implementation. Only atomic keys are allowed for encryption. Moreover, Truderung's approach cannot model some computations such as list mapping. This mapping aims to produce an encoded list $\{t'_1,..,t'_n\}_{k'}$ from an encoded list $\{t_1,..,t_n\}_k$ where, for each $i = 1,..,n$, the term t'_i is the result of applying some simple rewriting rule to the term t_i. Note that non-atomic keys can be handled by our verification procedure (to be presented in the following sections). [12] extend this model to handle XOR operator. Security can then be decided for a class of recursive protocols where principals are forbidden to XOR several messages (depending on messages) received from the network. [10] extend the Truderung approach to model freshness of nonces and keys more accurately.

Contribution. We have presented in [6] a synchronous model for parameterized protocols with bounded number of sessions, that can be viewed also as an extension of classical protocol models such as [22] to handle lists of messages whose length is given as a parameter. These protocols can be found in different domains. As example, we can cite web services protocols where lists of messages are commonly used. We can consider for instance, a basic action of an honest participant who received a list $m_1,..,m_n$ of messages whose length n is not known in advance and replies a message based on the informations he got:

$$\{\!|m_1|\!\}_k . \ldots . \{\!|m_n|\!\}_k \longrightarrow f(m_1 . \ldots . m_n)$$

In this step, the participant decrypts the different messages in the received list using the public key k of the sender. He then builds the message to be sent using the list $m_1,..,m_n$ of messages and a function f. Note that such protocols are used in Web Services where messages may contain unbounded lists of encrypted XML nodes.

Moreover, our model can handle in particular, group protocols that have an unbounded number of participants, and thus, admit this number as a parameter. Note that our running example for group protocols follows the basic action described above.

For this model, we have proposed a complete and correct set of inference rules that allows one to check the security of *well-tagged* protocols in this class in presence of an **active** intruder. We prove in this paper termination for our inference rules and satisfiability of the inferred solved constraints, for Well-Tagged protocols *with autonomous keys*. Thus, we show that our inference system provides a decision procedure for this class of protocols. Note that, since we consider parameterized protocols, in which the number of participants n is not fixed, if our procedure terminates without finding any attack, this proves that there is not any attack, for any value of the parameter n.

Moreover, the proposed rules can model **list mapping** and allow **non atomic keys**. The class of protocols for which we will prove decidability admits tagged messages. Tagging basically avoids some unifications between messages that could be exploited for attacks. Several works on protocols have considered tagging techniques on messages in order to enforce decidability. But these works ([4], [21]) do not consider group protocols, or protocols with unbounded lists. Moreover in our case tagging is limited to messages that contains indexed variables, that is variables to be instantiated by items of unbounded lists. The other messages do not need to be tagged.

Organization of the paper. We introduce in Section 1 our protocol and intruder model and the class of well-tagged protocols with autonomous keys for which we prove decidability. In Section 2, we detail the different constraints that are used to build constraints system that we consider and whose satisfiability is equivalent to the existence of attacks on a protocol. The set of simplification rules to reduce these constraints is given in Section 3. Finally, in Section 4, we present the algorithm for applying these rules on a constraints system modelling a protocol security problem. We introduce in the same section the different control characteristics that help to prove termination. We state then result of decidability of the insecurity problem for the class of Well-Tagged protocols with Autonomous Keys.

Motivating example

We consider here an example of a protocol that is inspired of the synchronous version of the Asokan-Ginzboorg group protocol [2] described in [6]. The following protocol aims at the establishment of a session key between a leader L and a simulator S that simulates n agents $a_1, \ldots, a_n$ having identical actions.

This protocol describes a simple establishment of a session key: the simulator sends the list of contributions of the different agents and the leader replies and sends its own contribution. These contributions are used to compute the

group key.

$$
\begin{array}{ll}
1. & S \longrightarrow L : (a_1 . \{\!|s_1|\!\}_k) . \ldots . (a_n . \{\!|s_n|\!\}_k) \\
2. & L \longrightarrow S : (\{\!|(s_1 . \ldots . s_n) . s'|\!\}_k) \\
 & \quad . \ldots . (\{\!|(s_1 . \ldots . s_n) . s'|\!\}_k) \\
3. & S \longrightarrow L : \left(a_1 . \{\!|s_1 . h((s_1 . \ldots . s_n) . s')|\!\}_{f((s_1 . \ldots . s_n) . s')}\right) \\
 & \quad . \ldots . \left(a_n . \{\!|s_n . h((s_1 . \ldots . s_n) . s')|\!\}_{f((s_1 . \ldots . s_n) . s')}\right)
\end{array}
$$

We assume that k is a symmetric key (a key encryption key) shared between the leader L and each participant a_i and thus the simulator S. Each agent a_i has its identity a_i, the two hash functions h and f and its contribution to the group key is s_i (nonce). The contribution of the leader to the group key is noted s'.

1. Protocol and intruder model

We follow the protocol model in [6] which is an extension of the protocol model [22] that deals with parametric lists (whose length is the parameter). They are constructed with a new operator denoted by $mpair(\_,\_)$. The intuition is that a *mpair* message represents a list of messages built with the same pattern.

Let $\mathscr{X}$ be a set of variables denoted by capital letters. Let $\mathscr{I}$ be a countable set of index variables. Let $\overrightarrow{\mathscr{X}}$ be a set of symbols represented by over-arrowed capital letters, disjoint from $\mathscr{X}$. Let $\mathscr{X}_{\mathscr{I}} = \left\{ Y_i \,|\, \overrightarrow{Y} \in \overrightarrow{\mathscr{X}} \text{ and } i \in \mathscr{I} \right\}$ be a countable set of (indexed) variables. Similarly, let $\mathscr{C}$ and $\overrightarrow{\mathscr{C}}$ be (disjoint) sets of symbols, represented by (over-arrowed) lower case letters, and let $\mathscr{C}_{\mathscr{I}} = \left\{ c_i \,|\, c \in \overrightarrow{\mathscr{C}} \text{ and } i \in \mathscr{I} \right\}$. Elements in $\mathscr{C}$ and $\mathscr{C}_{\mathscr{I}}$ are called constants. Note that $Y_i \in \mathscr{X}_{\mathscr{I}}$ can not be a member of $\mathscr{X}$. In this paper, terms can be (optionally) tagged by an index. That is, let $\overrightarrow{e} \in \overrightarrow{\mathscr{C}}$ be a symbol that we reserve for tagging operations only. Then a term is an element of $\mathscr{T}$ in the following language:

$$
\begin{array}{rcl}
\mathscr{T}_s & = & \{\!|\mathscr{T}|\!\}^p_{\mathscr{T}} \,|\, \{\!|\mathscr{T}|\!\}^s_{\mathscr{T}} \,|\, h(\mathscr{T}) \,|\, \langle \mathscr{T}, \mathscr{T} \rangle \,|\, mpair(\mathscr{I}, \mathscr{T}) \\
 & & |\, \mathscr{X} \,|\, \mathscr{X}_{\mathscr{I}} \,|\, \mathscr{C} \,|\, \mathscr{C}_{\mathscr{I}} \setminus \{e_i \,|\, i \in \mathscr{I}\} \\
\mathscr{T} & = & [e_i, \mathscr{T}_s] \,|\, \mathscr{T}_s \qquad \text{with } i \in \mathscr{I}
\end{array}
$$

where $\mathscr{T}_s$ is by definition the set of untagged terms. The operators $\{\!|\_|\!\}^p_\_$ and $\{\!|\_|\!\}^s_\_$ represent asymmetric and symmetric encryptions respectively, $\_.\_$ is a pairing operator and h is a hash function. We denote by $f \in H$ a hash function f. $mpair(i,t)$ is a symbolic representation of a list (or tuple) of terms, built from the common pattern t by iterating i along integers.

The translation function (see Definition 2) gives the semantic of the *mpair* operator, i.e. it's interpretation in the standard Dolev-Yao. This translation function replaces any *mpair* by a sequence of *pair* applications. The number of such applications is given as a parameter e to the translation function. The integer represents the common length of lists of terms represented by any $mpair(,)$.

Before giving the translation function in Defintion 2, we define two kind of Index-operations: replacements, used also in the inference rules over constraints.

Definition 1: [Index-Replacement δ, Index-Substitution τ] An Index-Replacement δ (resp. Index-Substitution τ) is an application from $\mathscr{I}$ to $\mathscr{I}$ (resp. to non-negative integers) that is extended to indexed variables and constants with $\delta(X_i) = X_{\delta(i)}$ and $\delta(c_i) = c_{\delta(i)}$ (resp. $\tau(X_i) = X_{\tau(i)}$ and $\tau(c_i) = c_{\tau(i)}$) and extended to terms and sets of terms by homomorphism.

We will use the notations $\delta_{i,j}$ (resp. $\tau_{i,j}$) to denote the replacement (resp. substitution) of $i \in \mathscr{I}$ by $j \in \mathscr{I}$ (resp. $j \in \mathbb{N}$). We also use $\delta^k_{i,j}$ to denote the replacement of $i \in \mathscr{I}$ by $j \in \mathscr{I}$ and the other indexes apart from i by $k \in \mathscr{I}$. This notation is extended to sets with $\delta^k_{Q,j}$ for some set $Q \subseteq \mathscr{I}$. $\delta^k_{Q,j}$ replaces every element of Q by j and the others by k.

We define now the translation function as follows:

Definition 2: [Translation of terms] Let $\mathscr{T}_{DY}$ be the set of terms without any $mpair(,)$. Given some integer e, the function $^{-e}$ from $\mathscr{T}$ to $\mathscr{T}_{DY}$ is defined as follows:

$$
\begin{array}{rcl}
\overline{mpair(i,t)}^e & = & \overline{\tau_{i,1}(t)}^e . \ldots . \overline{\tau_{i,e}(t)}^e, \text{ and} \\
\overline{f(s_1,..,s_k)}^e & = & f(\overline{s_1}^e,..,\overline{s_k}^e) \text{ for any } f \neq mpair
\end{array}
$$

We denote by signature $\mathscr{G}$ the set of operators in $\mathscr{T}_s$. To simplify the syntax, in the following we will write t^i instead of $[e_i,t]$, and call it a *tagged term*. We also omit the tag i of a term t^i, whenever the tag i is not relevant to the discussion. We denote by $\mathscr{T}_g$ the set of ground terms, i.e. any term $t \in \mathscr{T}$ with no variable in $\mathscr{X}$ or $\mathscr{X}_{\mathscr{I}}$ and no *mpair* symbol. Ground terms will be used to describe messages that are circulated in a protocol run. They are also called messages. Given a term t we denote by $Var(t)$ (resp. $Cons(t)$) the set of variables (resp. constants) occurring in t.

To represent a list of terms, we iterate the pairing operator $\_.\_$. For instance to represent a,b,c,d we can use the term $a . (b . (c . d))$ and we shall write this term in a shorthand: $a . b . c . d$. However we do not assume any associativity property of pairing.

A substitution σ assigns terms to variables. A ground substitution assigns ground terms to variables. The application of σ to a term t is written $t\sigma$. These notations are extended to sets of terms E in a standard way: $E\sigma = \{t\sigma | t \in E\}$.

The set of subterms of t is denoted by $Subterm(t)$. It is defined recursively as follows: If t is a variable or a constant then $Subterm(t) = \{t\}$. If $t = f(t_1,..,t_n)$ with $t \in \mathscr{G}$, then $Subterm(t) = \{t\} \cup \cup_{i=1}^n Subterm(t_i)$. If $t = u^i$ then $Subterm(t) = \{t\} \cup Subterm(u)$. We denote by $\leq$ the subterm relation on $\mathscr{T}$.

The set of tag-subterms of t is denoted by $Subterm^{\bullet}(t)$. It is defined recursively as follows: If t is a variable or a constant then $Subterm^{\bullet}(t) = \{t\}$. If $t = f(t_1,..,t_n)$ or $t = f(t_1,..,t_n)^i$ with $t \in \mathscr{G}$, then $Subterm^{\bullet}(t) = \{t\} \cup$

$\cup_{i=1}^{n} Subterm^{\cdot}(t_i)$. Note that u is *not* considered as a tag-subterm of u^i. We denote by $\leq^{\cdot}$ the tag-subterm relation on $\mathscr{T}$.

We define the relation $\leqslant_m$ over $\mathscr{T} \times \mathscr{T}$ as the smallest reflexive and transitive relation such that if $t = f(t_1,..,t_n)$ or $t = f(t_1,..,t_n)^j$ with $f \neq mpair$, then for all $i = 1,\ldots,m$ we have $t_i \leqslant_m t$. Note that $t \leqslant_m u$ implies $t \leq u$.

Finally, we define the set of indexes occurring in a term as follows:

Definition 3: [Term Indexes] Given a term $t \in \mathscr{T}$, we denote by $Var_{\mathscr{I}}(t)$ the set of indexes in t, recursively defined as follows:

- $Var_{\mathscr{I}}(mpair(i,t)) = Var_{\mathscr{I}}(X) = Var_{\mathscr{I}}(c) = \emptyset$, with $X \in \mathscr{X}$ and $c \in \mathscr{C}$;
- $Var_{\mathscr{I}}(X_i) = Var_{\mathscr{I}}(c_i) = \{i\}$, with $X_i \in \mathscr{X}_{\mathscr{I}}$ and $c_i \in \mathscr{C}_{\mathscr{I}}$;
- $Var_{\mathscr{I}}(f(t_1,..,t_n)) = Var_{\mathscr{I}}(t_1) \cup .. \cup Var_{\mathscr{I}}(t_n)$;
- and $Var_{\mathscr{I}}(t^i) = Var_{\mathscr{I}}(t) \cup \{i\}$ otherwise.

Note that, according to this recursive definition, term indexes do not cross the operator *mpair*. That is, for a term $u = mpair(i,t)$, even if t contains indexes, the set of term indexes of u is empty.

Example

The modelisation of our motivating example is given below:

$$\begin{array}{lcl} 1.\ S & \longrightarrow & L : mpair(i, a_i . \{\!|s_i|\!\}_k) \\ 2.\ L & \longrightarrow & S : mpair(i, \{\!|mpair(j,s_j) . s'|\!\}_k) \\ 3.\ S & \longrightarrow & L : mpair(i, a_i . \\ & & \quad \{\!|s_i . h(mpair(k,s_k) . s')|\!\}_{f(mpair(k,s_k).s')}) \end{array}$$

In this modelisation, each list of uniform messages $m_1,..,m_n$ is transformed into the message $mpair(i,m_i)$. Uniform messages refer to messages having the same pattern.

1.1. Protocol specification, intruder and attack

A protocol is given by a set of principals and a finite list of steps for each principal. We associate to each principal A a partially ordered finite set of steps represented by $(W_A, <_{W_A})$ where W_A are the indexes of steps of A and $<_{W_A}$ is a partial order over W_A. Each step is of the form $R_\iota \Rightarrow S_\iota$ where R_ι is an expected message by A and S_ι is his reply. *Init* and *End* are fixed messages used to initiate and close a protocol session. A protocol is then defined as $P = \{R_\iota \Rightarrow S_\iota | \iota \in J\}$ where J is a set of indexes of steps which is partially ordered. Our notion of correct execution of a protocol session (or *protocol run*) follows [22].

Moreover, we follow the intruder model of Dolev and Yao [8]. The actions of the intruder are simulated by a sequence of rewrite rules on sequences of messages. These rules are defined as follows. We note $\longrightarrow^*_{DY}$ their reflexive and transitive closure.

Composition rules
$a_1,\ldots,a_n \rightarrow a_1 . \ldots . a_n$
$a, K \rightarrow \{\!|a|\!\}^p_K$
$a, b \rightarrow \{\!|a|\!\}^s_b$
$h, t \rightarrow h(t)$ for $h \in H$
$t \rightarrow t^i$ for any $i \in \mathscr{I}$

Decomposition rules
$a_1 . \ldots . a_n \rightarrow a_1,\ldots,a_n$
$\{\!|a|\!\}^p_K, K^{-1} \rightarrow a$
$\{\!|a|\!\}^s_b, b \rightarrow a$
$t^i \rightarrow t$

Note here that we don't have intruder rule for the *mpair* operator since the translation function defined in Definition 2 gives the interpretation of this operator in the standard Dolev-Yao model and then the intruder has only to manage pairs of messages.

The notion of attack is the same as the one defined in [6] which is based on the predicate Dy. For a message t, a set of messages E and a set of keys K, $Dy(t,E,K)$ checks whether the message t can be constructed by the intruder from the set of known messages E without using any element of of K as decryption key.

Definition 4: [Attack] Given a protocol $P = \{R'_\iota \Rightarrow S'_\iota | \iota \in J\}$, a secret *Sec* and assuming the intruder has as initial knowledge S_0, an attack is described by a ground substitution σ, an integer e, and a correct execution order $\pi : J \longrightarrow 1,\ldots,k$ s.t. $\forall i = 1,\ldots,k$, we have:

$$\overline{R_i}^e \sigma \in Dy(\overline{S_0}^e, \overline{S_1}^e \sigma, \ldots, \overline{S_{i-1}}^e \sigma, \emptyset)$$
$$\overline{Sec}^e \in Dy(\overline{S_0}^e, \overline{S_1}^e \sigma, \ldots, \overline{S_k}^e \sigma, \emptyset)$$

where $R_i = R'_{\pi^{-1}(i)}$ and $S_i = S'_{\pi^{-1}(i)}$.

According to this definition, an execution corresponds to an attack if:

- this execution can be executed. Indeed, at each step of the execution, the intruder has to be able to forge the message expected by the participant from the knowledge he acquires throughout previous steps.
- at the end of this execution, the intruder must be able to forge the secret *Sec* from the knowledge acquired throughout all the execution.

Note that lists have always the same length e. Then, for group protocols, we assume that the group has always the same number of members.

1.2. Well-tagged protocols with autonomous keys

Unfortunately, the insecurity problem (i.e. the existence of an attack) is undecidable in the general case. This can be shown by encoding Post Correspondence Problem (PCP) with two letters. Note that this requires only atomic keys.

Definition 5: [PCP protocol] Let $J = \{(\alpha_1,\beta_2),..,(\alpha_p,\beta_p)\}$ be an instance of PCP on the alphabet $\{a,b\}$. We define the protocol specification $P(J)$ coding J as the following, with $\mathscr{C} = \{a,b,0,t,u\}$, $\mathscr{X} = \{Z\}$, $\overrightarrow{\mathscr{X}} = \left\{\overrightarrow{A},\overrightarrow{B},\overrightarrow{X},\overrightarrow{Y}\right\}$ and only one honest participant :

1. $Init \Rightarrow a, b, 0, \{\!|0.0|\!\}_t$
2. $mpair(i,A_i.B_i) \Rightarrow mpair(i,\{\!|A_i.B_i|\!\}_t)$
3. $mpair(i,\{\!|X_i.Y_i|\!\}_t) \Rightarrow mpair(i,\{\!|\alpha_1(X_i).\beta_1(Y_i)|\!\}_u),$ $..mpair(i,\{\!|\alpha_p(X_i).\beta_p(Y_i)|\!\}_u)$
4. $mpair(i,\{\!|A_i.B_i|\!\}_u), \{\!|Z.Z|\!\}_u \Rightarrow Sec$

At Step 1, we provide the alphabet as constants to the intruder, as well as a termination symbol 0. At Step 2, the intruder is asked to construct a list (or *mpair*) of pairs of words over a,b. The following of the protocol will consist in testing if each of these pairs can be obtained from an other one increased with one of the PCP words (α_j,β_j). Since the initial pair of empty words represented by $\{\!|0.0|\!\}_t$, cannot be obtained this way, it is provided to the intruder separately at Step 1. We fix the intruder choice at step 2 by encrypting it with key t. Then, at Step 3 the intruder selects some of the pairs of words encrypted by t that he got at step 1 and 2, and receives each of them back extended with one of the pairs of words of the PCP instance and encrypted by u. We expect the intruder to select all of the pairs he has chosen at Step 2 minus the longest one, replaced by the pair of empty words. Finally, at Step 4 we perform two verifications: first, we test that for any pair of words chosen at Step 2, there exists an extended pair of words received at Step 3, i.e. by recursion that each pair of terms chosen at Step 2 is a concatenation of words of J, the instance of PCP from which we build this protocol; second, we test that one of the extended pair of words is a solution to PCP problem, i.e. a pair of identical words.

Theorem 1: Let J be an instance of PCP. J has a solution iff $P(J)$ has an attack on *Sec*.

The proof of this theorem follows from Lemmas 1 and 2.

Lemma 1: If for some integer n there is a run of $P(J)$, then J has a solution.

Proof: According to the protocol definition, all we need to do is to backtrack recursively the creation of extended pairs at Step 2 and 3: for any term $\{\!|a.b|\!\}_u$ known by the intruder, including $\{\!|A_j.B_j|\!\}_u$ for any $j \in 1..n$ or $\{\!|Z.Z|\!\}_u$, there exists $i \in 1..n$ and $k \in 1..p$ such that $a = \alpha_k(X_i)$ and $b = \beta_k(Y_i)$. Therefore, either $X_i = Y_i = 0$, or there exists $i' \in 1..n$ such that $a = \alpha_k(A_{i'})$ *and* $b = \beta_k(B_{i'})$, and the intruder knows $\{\!|A_{i'}.B_{i'}|\!\}_t$. By iteration on $A_{i'}$and $B_{i'}$, starting from $\{\!|Z.Z|\!\}_u$, and since $|a.b| > |A_{i'}.B_{i'}|$, it appears that there exists a list $[j_1,..,j_r]$ of indexes in $1..p$ such that $\alpha_{j_1}(\alpha_{j_2}(..\alpha_{j_r}(0)..)) = \beta_{j_1}(\beta_{j_2}(..\beta_{j_r}(0)..))$, i.e. $\alpha_{j_1}..\alpha_{j_r} = \beta_{j_1}..\beta_{j_r}$. □

Lemma 2: If there is a solution to J, there exists n such that $P(J)$ admits a run.

Proof: Let $[j_1,..,j_r]$ be a list of indexes in $1..p$ such that $\alpha_{j_1}..\alpha_{j_r} = \beta_{j_1}..\beta_{j_r}$. We choose $n = r$. We also choose the following values for variables A_i and B_i :

- $\sigma(A_i) = \alpha_{j_i}(\sigma(A_{i+1}))$ and $\sigma(B_i) = \beta_{j_i}(\sigma(B_{i+1}))$ for any $i \in 1..n-1$;
- $\sigma(A_n) = \alpha_{j_n}(0)$ and $\sigma(B_n) = \beta_{j_n}(0)$.

This is the set of pairs of words chosen by the intruder at step 2. Then, at step 3 we chose :

- $\sigma(X_1) = \sigma(Y_1) = 0$;
- $\sigma(X_i) = \sigma(A_i)$ and $\sigma(Y_i) = \sigma(B_i)$, for any $i \in 2..n$.

i.e. we keep all the pairs of words chosen at step 2 except that we replace the longest (final) one, A_1,B_1 by the pair of empty words. We now easily pass the tests of step 4, since:

- $\forall i \in 1..n-1$, $\exists k \in 1..p$, $\exists j \in 1..n$ s.t. $\sigma(A_i) = \alpha_k(\sigma(X_j))$ and $\sigma(B_i) = \beta_k(\sigma(Y_j))$;
- $\sigma(A_n) = \alpha_{j_n}(0)$ and $\sigma(B_n) = \beta_{j_n}(0)$;
- $\sigma(A_1) = \alpha_{j_1}(..\alpha_{j_n}(0)..) = \beta_{j_1}(..\beta_{j_n}(0)..) = \sigma(B_1)$.

According to our choices of variables A_i, B_i, X_i and Y_i, this is true for $k = j_i$ and $j = i+1$ at least. □

Consequently, finding an attack to PCP with two letters is no more difficult than finding a run in a parameterized protocol as defined in this paper. Since finding a run is no more difficult than finding an attack (the secret can be released at the end), it follows that the insecurity problem of parameterized protocol without further restrictions is undecidable.

We will therefore introduce the class of Well-tagged protocols with autonomous keys for which decidability is expected. To do this, we first introduce the notion of autonomy:

Definition 6: [Autonomy] A term $mpair(i,u)$ is autonomous when $Var_{\mathscr{I}}(u) \subseteq \{i\}$. A term $t \in T_{DY}$ is autonomous if $\#Var_{\mathscr{I}}(t) \leq 1$ and $\forall t' < t$, t' is autonomous. A protocol $\mathscr{P} = \{R_\iota \Rightarrow S_\iota | \iota \in J\}$ is autonomous iff for all $\iota \in J$, R_ι and S_ι are autonomous and $Var_{\mathscr{I}}(R_\iota) = \emptyset$ and $Var_{\mathscr{I}}(S_\iota) = \emptyset$.

Example 1: As examples of autonomous terms, we have the term $mpair(i,a_i.mpair(j,\{\!|c_j|\!\}_k))$. Nevertheless, the term $t = mpair(i,mpair(j,\{\!|a_i|\!\}_{c_j}))$ is not autonomous.

We introduce now the notion of Well-Tagged protocols. The idea underlying the tagging of variables is to add enough information on terms in *mpair* so that the protocol cannot be used to test or guarantee relations between elements of the same *mpair*, such as $\forall i$, $\exists i'$ s.t. $X_i = f(X_{i'})$.

Definition 7: [Well-Tagged protocols] A protocol $\mathscr{P} = \{R_\iota \Rightarrow S_\iota | \iota \in J\}$ is Well-Tagged iff:

1) $\forall \iota \in J$, $\forall X_i \in \mathscr{X}_{\mathscr{I}} \cap Subterm^{\bullet}(R_\iota) \cap \bigcup_{\iota' < \iota} Subterm^{\bullet}(R_{\iota'})$, X_i is tagged;
2) $\forall \iota \in J$, $\forall X_i \in \mathscr{X}_{\mathscr{I}} \cap Subterm^{\bullet}(S_\iota)$, X_i is tagged;
3) $\forall \iota \in J$, $\forall t = f(s_1,..,s_k) \in Subterm^{\bullet}(R_\iota \cup S_\iota)$ with $f \neq mpair$, if $\exists j = 1..k$ s.t. s_j is tagged, then t is tagged too with the same tag;

4) $\forall \iota \in J$, $\forall t \in Subterm^{\star}(R_\iota)$ tagged, $\forall X_i \leq t$ where $X_i \in \mathcal{X}_{\mathscr{I}}$, X_i is tagged.
5) $\mathscr{P}$ is autonomous.

In this definition, Conditions 1 and 2 state that any indexed variable of the protocol must be tagged, except for its first occurrence w.r.t. the partial step ordering. Moreover, Condition 3 (when combined with Conditions 1 and 2) states that, for any subterm t of the protocol, if an indexed variable is accessible from t by decompositions without opening any *mpair*, then t must be tagged. Note that as a consequence of mpair autonomy, an indexed variable X_i can only appear tagged by its index (as in ${X_i}^i$) or untagged. Condition 4 states that every indexed variable subterm of a tagged term of R_ι is tagged.

Example 2: To clarify the definition of the class of Well-Tagged protocols, we give here examples of protocols that are either well-tagged or not.

- Let $P = \{step_1, step_2\}$ where, for some terms t,t', a key k and an indexed variable X_i, we have $step_1 = mpair(i,X_i) \Rightarrow t$ and $step_2 = mpair(i,\{\!|X_i|\!\}_k) \Rightarrow t'$. Then, P is not well-tagged protocol since condition 1 is violated because the variable X_i must be tagged in the step $step_2$ which is not the case here.
- Let $t \Rightarrow mpair(i,X_i)$ be a step of a protocol P for some term t and an indexed variable X_i. Then, P is not well-tagged protocol since condition 2 is not satisfied because X_i has to be tagged whereas it is not tagged here.
- If $mpair(i,t'.{X_i}^i) \Rightarrow t$ or $t \Rightarrow mpair(i,t'.{X_i}^i)$ is a step of a protocol P for some terms t,t' and an indexed variable X_i, then P is not well-tagged. Indeed, condition 3 is not satisfied since $t'.{X_i}^i$ must be tagged.
- Let $mpair(i,{\{\!|X_i|\!\}_k}^i) \Rightarrow t$ be a step of a protocol P. Then, P is not well-tagged since condition 4 is not satisfied because X_i must be tagged which is not the case here.
- Let $P = \left\{ mpair(i,X_i) \Rightarrow mpair(i,{\{\!|{X_i}^i|\!\}_k}^i) \right\}$ be a protocol where X_i is an indexed variable and k is a key. Then, P is well-tagged.

Finally, we introduce the notion of protocols with autonomous keys for which decidability will be proved. This notion aims at preventing the case where index variables are generated from keys.

Definition 8: [Protocols with autonomous keys] A protocol $\mathscr{P} = \{R_\iota \Rightarrow S_\iota | \iota \in J\}$ is called with autonomous keys iff $\forall \iota \in J\ \forall t \in Subterm(R_\iota \cup S_\iota)$ s.t $t = \{\!|u|\!\}_v$, $Var_{\mathscr{I}}(v) = \emptyset$.

Example 3: We give here some examples of terms that have autonomous keys and other terms that have not this feature. Terms $\{\!|t|\!\}_{s_i}$ and $\{\!|t|\!\}_{X_i}$ have not autonomous keys. Terms $\{\!|t|\!\}_{mpair(i,s_i)}$ and $\{\!|t|\!\}_k$ have autonomous keys.

2. Constraints for verification of well-tagged protocols with autonomous keys

We will use a symbolic constraints system to represent all runs of a protocol given a step ordering. This system uses (universal or existential) quantifiers on index variables and includes an (implicit) universal quantification on the number n of elements in any *mpair*. Before defining our constraints system, some basic notions have to be introduced. For terms s,s' (resp. sets of terms E,E') we note $s \sim s'$ (resp. $E \sim E'$) if they are equal once we erase their tags. We define now the relation for accessible terms.

Definition 9: [Relation $\leqslant_E^L$ for accessible subterms] We consider a relation $\_ \leqslant_{\bar{\mathscr{T}}} \_$ on $\mathscr{T} \times 2^{\mathscr{T}} \times 2^{\mathscr{T}} \times \mathscr{T}$. We write $s \leqslant_E^L t$ for s,t terms in $\bar{\mathscr{T}}$ and E and L finite subsets of $\mathscr{T}$. Note that this can be used for $\mathscr{T}_s$ too. This relation is defined as the smallest relation such that:

$t \leqslant_\emptyset^\emptyset t \quad \forall t \in \mathscr{T}$
$s \leqslant_E^L t$ iff $s' \leqslant_{E'}^{L'} t'$ where $s \sim s', t \sim t', E \sim E', L \sim L'$
If $m \leqslant_{E'}^L t$ and $E' \subset E$ then $m \leqslant_E^L t$
If $\{\!|m|\!\}_k^p \leqslant_E^L t$ then $m \leqslant_{E,k^{-1}}^{L'} t$ where $L' = L \cup \{\!|m|\!\}_k^p\}$
If $\{\!|m|\!\}_b^s \leqslant_E^L t$ then $m \leqslant_{E,b}^{L'} t$ where $L' = L \cup \{\!|m|\!\}_b^s\}$
If $t_1.\ldots.t_n \leqslant_E^L t$ then $\forall i \leq n, t_i \leqslant_E^{L'} t$ where $L' = L \cup \{t_1.\ldots.t_n\}$

We write $u \leqslant_E t$ when $u \leqslant_E^L t$ for some L. Note that by construction, $u \leqslant_E t$ implies $u \in Subterm(t)$. We say that u is a subterm of t that is accessible, i.e. can be obtained from t by decompositions using keys in E. The set L denotes the set of intermediate terms between t and u including t. For simplicity, we note $\leqslant_{b_1,..,b_k}$ instead of $\leqslant_{\{b_1,..,b_k\}}$. We define also the set of strict accessible terms by $s <_F t$ (resp. $s <_F^L t$) if $s \leqslant_F t$ (resp. $s \leqslant_F^L t$) and $s \neq t$. Given $t \leqslant_F^L u$ or $t <_F^L u$, we call length of $t \leqslant_F^L u$ (resp $t <_F^L u$) the number of elements of L.

Example 4: Examples of $\leqslant_E^L$ and $<_E^L$.

As a simple example of $\leqslant_E^L$, we can cite $t \leqslant_\emptyset^\emptyset t$. For $\leqslant_E^L$, we can mention : $t <_{\{k,k'\}}^{\{\{t'.\{t\}_{k'}\}_k, t'.\{t\}_{k'}, \{t\}_{k'}\}} \{\!|t'.\{\!|t|\!\}_{k'}|\!\}_k$. The length of the last one is the number of elements of $\{\{\!|t'.\{\!|t|\!\}_{k'}|\!\}_k, t'.\{\!|t|\!\}_{k'}, \{\!|t|\!\}_{k'}\}$ (3) which represents the number of intermediate terms between $\{\!|t'.\{\!|t|\!\}_{k'}|\!\}_k$ and t including $\{\!|t'.\{\!|t|\!\}_{k'}|\!\}_k$.

Before defining constraints systems, we need to introduce elementary and negative constraints that compose this system. Constraints of the system use the environment defined as follows :

Definition 10: [Environment] We call an *environment* a finite set of equalities $X = u$ whose left-hand sides are variables ($X \in \mathcal{X} \cup \mathcal{X}_{\mathscr{I}}$), with an existential quantifier prefix, i.e. a finite list of index variables quantified existentially $\exists j_1,..,\exists j_p$, that we note in short $\exists R'$. We usually denote the environment by $\mathscr{E}$.

An environment of a block B contains the set of master constraints of all the blocks and the sub-master constraints of B. Elementary constraints are defined in Definition 11.

Definition 11: [Elementary Constraint] An elementary constraint is an expression $(t = t')$, $(t \in Forge(E, \mathscr{K}))$, $(t \in Forge_c(E, \mathscr{K}))$, $(t \in Sub(t', E, \mathscr{E}, \mathscr{K}))$, or $(t \in Sub_d(t', E, \mathscr{E}, \mathscr{K}))$ with an environment $\mathscr{E}$, $t, t' \in \mathscr{T}$ and $E \subset \mathscr{T}$.

An elementary constraint represents a basic relation on terms. Indeed, $t \in Forge(E, \mathscr{K})$ if the term t is derivable from the knowledge E without using elements $\mathscr{K}$ as decryption keys; $t \in Forge_c(E, \mathscr{K})$ if t is derived by composition; $t \in Sub(t', E, \mathscr{E}, \mathscr{K})$ if t is an accessible subterm from t' with knowledge E with none of the intermediate terms between t and t' in $\mathscr{K}$, and this modulo replacements using equations of $\mathscr{E}$; $t \in Sub_d(t', E, \mathscr{E}, \mathscr{K})$ if t is accessible by decomposition of t', also modulo replacements using $\mathscr{E}$; and $t = t'$ if t and t' are equal. Formally, we define the set of solutions of elementary constraints in Definition 12. The set of solutions of a constraint S, denoted by $[\![S]\!]^e_\tau$ where e is a value of n and τ is an Index-Substitution is a set of ground substitutions to be defined in the following. We define $\mathscr{G}S$ to be the set of all ground substitutions.

Definition 12: [Solutions of an Elementary Constraint]

$$[\![t = t']\!]^e_\tau = \left\{\sigma \in \mathscr{G}S \mid \overline{t}^e \tau\sigma = \overline{t'}^e \tau\sigma\right\}$$

$$[\![t \in Forge(E, \mathscr{K})]\!]^e_\tau = \left\{\sigma \in \mathscr{G}S \mid \overline{t}^e \tau\sigma \in Dy(\overline{E}^e \tau\sigma, \overline{\mathscr{K}}^e \tau\sigma)\right\}$$

$$[\![t \in Forge_c(E, \mathscr{K})]\!]^e_\tau = \left\{\sigma \in \mathscr{G}S \mid \overline{t}^e \tau\sigma \in Dy_c(\overline{E}^e \tau\sigma, \overline{\mathscr{K}}^e \tau\sigma)\right\}$$

$$[\![t \in Sub(w, E, \mathscr{E}, \mathscr{K})]\!]^e_\tau = \lrcorner\sigma \in \mathscr{G}S \mid \exists u\ \exists F, L \text{ s.t.}$$
$$u \leqslant^L_F \overline{w}^e \tau,$$
$$F\sigma \subseteq Dy(\overline{E}^e \tau\sigma, \overline{\mathscr{K}}^e \tau\sigma),$$
$$L\sigma \cap \overline{\mathscr{K}}^e \tau\sigma = \emptyset, \text{ and}$$
$$\text{either } u\sigma = \overline{t}^e \tau\sigma$$
$$\text{or } \exists v, \delta, k,\ \tau' \text{ s.t}$$
$$\begin{cases} \begin{cases} \text{either } u \in \mathscr{X},\ (u = v) \in \mathscr{E}, \\ \quad \delta = \emptyset, \text{ and } \tau' = \tau \end{cases} \\ \begin{cases} \text{or } \exists \overrightarrow{Z} \in \overrightarrow{\mathscr{X}}, i, j \in \mathscr{I} \text{ s.t} \\ \quad u = Z_i\tau',\ (Z_j = v) \in \mathscr{E}, \\ \quad \delta = \delta^k_{j,i}, \tau \subseteq \tau' \text{ and} \\ \quad \mathscr{D}om(\tau') = \mathscr{D}om(\tau) \cup \{k, i\} \end{cases} \\ k, i \notin Var_\mathscr{I}(\{t, w, \mathscr{E}\}), \\ u\sigma \notin Dy_c(\overline{E}^e \tau\sigma, \overline{\mathscr{K}}^e \tau\sigma) \\ u\sigma = \overline{v}^e \delta\tau'\sigma, \text{ and} \\ \sigma \in [\![t \in Sub(v\delta, E, \mathscr{E}, \mathscr{K})]\!]^e_{\tau'} \end{cases} \lrcorner$$

$[\![t \in Sub_d(w, E, \mathscr{E}, \mathscr{K})]\!]^e_\tau$ is defined in a similar way as $[\![t \in Sub(w, E, \mathscr{E}, \mathscr{K})]\!]^e_\tau$ with the difference that, for the first case (when $u\sigma = \overline{t}^e \tau\sigma$), we have $u <^L_F \overline{w}^e \tau$.

In this definition, the set of solutions of a constraint of type $t \in Forge(E, \mathscr{K})$, given an index substitution τ and an integer e, is the set of close substitutions σ such that $\overline{t}^e \tau\sigma$ is derivable from the set $\overline{E}^e \tau\sigma$ without using any element of $\overline{\mathscr{K}}^e \tau\sigma$ as a decryption key.

For the set of solutions of a constraint of type $t \in Sub(w, E, \mathscr{E}, \mathscr{K})$, let τ be an index substitution and e an integer. For a set of keys $\mathscr{K}$ and a set of knowledges E (in the definition, we have respectively $\overline{\mathscr{K}}^e \tau\sigma$ and $\overline{E}^e \tau\sigma$), we say that a term u is accessible from a term v if u is a subterm of v while generating all the keys between u and v from $\overline{E}^e \tau\sigma$ without using any key of $\overline{\mathscr{K}}^e \tau\sigma$.

The set of solutions of $t \in Sub(w, E, \mathscr{E}, \mathscr{K})$ is then the set of close substitutions σ such that :

- either $\overline{t}^e \tau\sigma$ is accessible from $\overline{w}^e \tau\sigma$,
- or there exists a variable (indexed or not) subterm accessible from w such that, by replacement of this variable using a representative equation from the environment $\mathscr{E}$, our term $\overline{t}^e \tau\sigma$ becomes accessible from the chosen value of this variable.

Example 5: Examples of solutions of elementary constraints.

We use the notation $[X \leftarrow c]$ to define a substitution that assigns the value c to the variable X. Indexed terms denote indexed variables (capital letters) or indexed constants (otherwise). In Figure 1, we present lists the semantic of remarkable elementary constraints. They are quite numerous, but try to cover all major cases.

We define negative constraints as follows :

Definition 13: [Negative Constraint] A negative constraint is an expression $(\forall i\, X_m \neq u)$ or $(X_m \notin Forge_c(E, \mathscr{K}))$ with $X_m \in \mathscr{X}_\mathscr{I}$, $u \in \mathscr{T}$, $E, E' \subset \mathscr{T}$ and $i \in Var_\mathscr{I}(u)$.

The first constraint aims at eliminating among the solutions (close substitutions) those that assign to the variable X_m a value among the values of u and this, for all the values of the index variable in u. The second constraint permits to eliminate solutions that assign to a variable X_m a value forgeable from the set of knowledges E. The set of solutions of negative constraints is given in Definition 14.

Definition 14: [Solutions of a Negative Constraint]

$$[\![X_m \notin Forge_c(E, \mathscr{K})]\!]^e_\tau = \mathscr{G}S \setminus [\![X_m \in Forge_c(E, \mathscr{K})]\!]^e_\tau$$
$$[\![(\forall i\, X_m \neq u)]\!]^e_\tau = \mathscr{G}S \setminus \bigcup_{x=1\ldots e} [\![X_m = u]\!]^e_{[i \leftarrow x], \tau}$$

We can now describe our constraints system by blocks in the following way:

Definition 15: [Constraints System] First, we define a constraints block B as a conjunction of constraints together with an environment $\mathscr{E}$: $B = (ctr_1 \wedge \ldots \wedge ctr_l, \mathscr{E})$. We will sometimes handle blocks as set of elementary or negative constraints for ease of notations. For instance we write $c \in B$ to express that the constraint c is a conjunct of B.

$$[\![\{\!\{X\}\!\}_k = \{\!\{c_i\}\!\}_Y]\!]^e_{\tau,i\leftarrow 1} = [X \leftarrow c_1, Y \leftarrow k]$$

$$[\![f(X) \in Forge(\{\{\!\{f(c)\}\!\}_k, k\}, \{\{k\}\})]\!]^e_\tau = \emptyset$$

$$\left|\left[\{\!\{f(X_i)\}\!\}_k \in Forge_c(\left\{f(c), k, \{\!\{f(r_j)\}\!\}_k\right\}, \emptyset) \right]\right|^e_{\tau,i\leftarrow 1, j\leftarrow 2} = [X_1 \leftarrow c]$$

$$[\![\{\!\{X\}\!\}_k \in Sub(\{\!\{c \centerdot \{\!\{r_i\}\!\}_k\}\!\}_k, \emptyset, \emptyset, \emptyset)]\!]^e_{\tau,i\leftarrow 1} = [X \leftarrow c \centerdot \{\!\{r_i\}\!\}_k]$$

$$[\![\{\!\{X\}\!\}_k \in Sub(\{\!\{c \centerdot \{\!\{r_i\}\!\}_k\}\!\}_k, \{k', \{\!\{k\}\!\}_{k'}\}, \emptyset, \{k'\})]\!]^e_{\tau,i\leftarrow 1} = [X \leftarrow c \centerdot \{\!\{r_i\}\!\}_k]$$

$$[\![\{\!\{X\}\!\}_k \in Sub_d(\{\!\{c \centerdot \{\!\{r_i\}\!\}_k\}\!\}_k, \{k', \{\!\{k\}\!\}_{k'}\}, \emptyset, \{k'\})]\!]^e_{\tau,i\leftarrow 1} = \emptyset$$

$$[\![\{\!\{X\}\!\}_k \in Sub(\{\!\{c \centerdot \{\!\{r_i\}\!\}_k\}\!\}_k, \{k', \{\!\{k\}\!\}_{k'}\}, \emptyset, \emptyset)]\!]^e_{\tau,i\leftarrow 1} = [X \leftarrow c \centerdot \{\!\{r_i\}\!\}_k] \cup [X \leftarrow r_1]$$

$$[\![\{\!\{X\}\!\}_k \in Sub_d(\{\!\{c \centerdot \{\!\{r_i\}\!\}_k\}\!\}_k, \{k', \{\!\{k\}\!\}_{k'}\}, \emptyset, \emptyset)]\!]^e_{\tau,i\leftarrow 1} = [X \leftarrow r_1]$$

$$[\![\{\!\{X\}\!\}_k \in Sub_d(\{\!\{c \centerdot Y\}\!\}_k, \{k\}, \{(Y = r)\}, \emptyset)]\!]^e_\tau = [X \leftarrow r]$$

$$\left|\left[\{\!\{X\}\!\}_k \in Sub_d(\{\!\{c \centerdot Y_i\}\!\}_k, \{k\}, \left\{(Y_j = r_j)\right\}, \emptyset) \right]\right|^e_{\tau,i\leftarrow 1} =$$
$$\left|\left[\{\!\{X\}\!\}_k \in Sub_d(\{\!\{c \centerdot r_i\}\!\}_k, \{k\}, \left\{(Y_j = r_j)\right\}, \emptyset) \right]\right|^e_{\tau,i\leftarrow 1} = [X \leftarrow r_1]$$

Let τ' be s.t $\tau, i \leftarrow 1 \subset \tau'$ and $\tau'(l) = 2$ for $l \notin \mathscr{D}om(\tau)$:

$$\left|\left[\{\!\{X\}\!\}_k \in Sub_d(\{\!\{c \centerdot Y_i\}\!\}_k, \{k\}, \left\{(Y_j = r_m)\right\}, \emptyset) \right]\right|^e_{\tau,i\leftarrow 1} =$$
$$\left|\left[\{\!\{X\}\!\}_k \in Sub_d(\{\!\{c \centerdot r_l\}\!\}_k, \{k\}, \left\{(Y_j = r_m)\right\}, \emptyset) \right]\right|^e_{\tau'} = [X \leftarrow r_2]$$

Figure 1. Examples of solutions of elementary constraints

We can now define the constraints system that we will use to represent protocol runs. Given two finite lists of index variables $Q = i_1, \ldots, i_k$ and $R = j_1, \ldots, j_l$, we write the quantifier prefix $\forall i_1 \cdots \forall i_k \exists j_1 \cdots \exists j_l$ in short: $\forall Q\, \exists R$. A constraints system, denoted by S, is a disjunction of blocks with a quantifier prefix: $S = \forall Q\, \exists R\, (B_1 \vee \ldots \vee B_p)$

The idea will be to use a constraints system based on blocks to represent all possible ways the intruder can construct a list of terms represented by an *mpair*. Roughly, there will be one block in the system for each way. The set of solutions of a constraints system is given in Definition 16.

Definition 16: [Solutions of the Constraints System] Consider a constraints system S, B_i, for $i = 1 \ldots p$ (blocks of S) and $ctr_{i,j}$, for $j = 1 \ldots l_i$ (constraints of the block B_i) given in Definition 15. The set of solutions of a constraints system CS is defined inductively using the following cases:

$$[\![\forall i\, S]\!]^e_\tau = \cap_{x=1,\ldots,e} [\![S]\!]^e_{[i\leftarrow x],\tau} \qquad [\![S]\!]^e_\tau = \bigcup_{i=1\ldots p} [\![B_i]\!]^e_\tau$$

$$[\![\exists i\, S]\!]^e_\tau = \cup_{x=1,\ldots,e} [\![S]\!]^e_{[i\leftarrow x],\tau} \qquad [\![B_i]\!]^e_\tau = \bigcap_{j=1\ldots l_i} \left|[\![ctr_{i,j}]\!]\right|^e_\tau$$

In this definition, the set of solutions of a constraints system (without quantifiers), modulo some index substitution, is the set of solutions satisfying at least one the blocks of this system. In a similar way, the set of solutions of a constraints block (without quantifiers), modulo some index substitution, is the set of solutions satisfying all the constraints composing this block. Moreover, the set of solutions of a constraints system with an universal quantifier is the set of solutions satisfying this system for all the possible values that can assign the index variable of this quantifier. In the same way, the set of solutions of a constraints system with an existential quantifier is the set of solutions satisfying this system for at least one of the possible values that can assign the index variable of this quantifier. Here are some examples.

Example 6: Examples of solutions of a constraints system.

$$\begin{aligned}
&\left|[\![\forall i\ ((X_i = c) \wedge (X_j = r))]\!]\right|^e_{\tau,j\leftarrow 1} \\
&= \cap_{x=1,\ldots,e} \left|[\![((X_i = c) \wedge (X_j = r))]\!]\right|^e_{\tau,j\leftarrow 1,i\leftarrow x} \\
&= \left|[\![(X_j = r)]\!]\right|^e_{\tau,j\leftarrow 1,i\leftarrow x} \cap \cap_{x=1,\ldots,e} \left|[\![(X_i = c)]\!]\right|^e_{\tau,j\leftarrow 1,i\leftarrow x} \\
&= \emptyset
\end{aligned}$$

$$\begin{aligned}
&\left|[\![\exists i\ ((X_i = c) \wedge (X_j = r))]\!]\right|^e_{\tau,j\leftarrow 1} \\
&= \cup_{x=1,\ldots,e} \left|[\![((X_i = c) \wedge (X_j = r))]\!]\right|^e_{\tau,j\leftarrow 1,i\leftarrow x} \\
&= [X_1 \leftarrow r, X_2 \leftarrow c] \cup \ldots \cup [X_1 \leftarrow r, X_e \leftarrow c]
\end{aligned}$$

Notation 1 (labels of constraints): Blocks are extended to admit labeled constraints as defined in [6].

A constraint ctr may be equipped with a label $(ctr)^m$ or $(ctr)^{sm}$ or $(ctr)^f$ or also $(ctr)^d$ to denote respectively a master constraint or a submaster constraint or a final constraint or a dull constraint.

The two first labels m and sm will be used to select representative values for an indexed and non-indexed variables. For example, we will prove that we have exactly one master constraint for every indexed variable in each block, and we will use master or sub master constraints to instantiate variables when needed;

For $\overrightarrow{X} \in \overrightarrow{\mathscr{X}}$, a constraint system S, we denote by $\mathscr{M}(S,\overrightarrow{X})$ the set of master constraints in S related to $\overrightarrow{X}$, i.e. constraints of the form $(X_i \in Forge_c(E,K))^m$ or $(X_j = u)^m$ where $i, j \in Q$. When S is clear from the context, $\mathscr{M}(S,\overrightarrow{X})$ will be noted in short $\mathscr{M}(\overrightarrow{X})$. In the same way, for $X \in \mathscr{X}$, a constraints block B, we denote by $\mathscr{SM}(B,X)$ the set of submaster constraints related to X, i.e. constraints of the form $(X = u)^{sm}$. This notion is extended to deal with sets of variables : for a set of non indexed variables W, $\mathscr{SM}(B,W)$ denotes $\bigcup_{X\in W} \mathscr{SM}(B,X)$. The formal definition of these three sets of constraints can be found in [5].

The two last labels f and d will be used to avoid any further rewriting on some constraint. A dull constraint prevents further rewriting on some constraint unless the set of master constraints change by the addition of new master constraint or the modification of an existing one. In this case, dull labels are omitted from constraints by application of a function named *awake*. For example, $awake((ctr)^d) = ctr$. This function is applied to all the constraints system. The formal definition of this function can be found in [5]. We introduce also the notation $(ctr)^{\bullet}$ to refer to labelled or non labelled constraint.

The solutions of labelled constraints are the solutions of the constraints obtained by removing labels.

Example 7: We give as example the following system of constraints, where $\overrightarrow{X} \in \overrightarrow{\mathscr{X}}$, $Y \in \mathscr{X}$, $\overrightarrow{c} \in \overrightarrow{\mathscr{C}}$ and $b \in \mathscr{C}$:

$$S = \forall\{i,l,m,\}\exists\{o,p,r,s,k'\}\ B_1 \vee B_2 \vee B_3 \vee B_4 \vee B_5$$

with

$$\begin{aligned}
B_1 &= ((X_i \in Forge_c(E,\mathscr{K}))^m \wedge (Y = c_s)^{sm})\\
B_2 &= ((X_l = \{\!|c_l|\!\}_b)^m \wedge Y \in Forge_c(E',\mathscr{K}'))\\
B_3 &= ((Y = c_r)^{sm} \wedge (X_m = c_p)^m \\
&\quad \wedge(\{\!|Y|\!\}_b = \{\!|c_o|\!\}_b) \wedge (X_o = \{\!|c_o|\!\}_b)^f)\\
B_4 &= ((Y = c_r)^{sm} \wedge (X_m = c_p)^m \\
&\quad \wedge(\{\!|Y|\!\}_b = c_{k'}) \wedge (X_o = c_{k'})^f)\\
B_5 &= ((Y = c_r)^{sm} \wedge (X_m = c_p)^m \\
&\quad \wedge\{\!|Y|\!\}_b \in Forge_c(E,\mathscr{K}) \wedge (X_0 = \{\!|Y|\!\}_b)^d \\
&\quad \wedge(X_o \neq c_o) \wedge (\forall k X_o \neq c_k))
\end{aligned}$$

3. Normalization of a Constraints System

In this section we present the rules applied in the normalization function over constraints systems. The result of applying a rule is put in disjunctive normal form and existential quantifiers are moved up to the prefix of the system using first order logic. These rules are organized in six groups $G_1, \ldots, G_6$. In this section, we give a summary and some examples of rules of these groups. The whole set of rules can be found in [5].

G_1 aims at maintaining syntactic properties over a constraints system. Some rules handle labelling of master constraints by adding new labels or transferring existing ones. We call them respectively the labelling and the label transfer rules. Labelling rules can add master and submaster labels if the block considered has not yet such constraints. These rules are defined as follows:

$$\begin{aligned}
B \wedge X = u \quad &\longrightarrow \quad B \wedge (X = u)^{sm}\\
&\qquad \text{where } \mathscr{SM}(B,X) = \emptyset
\end{aligned}$$

$$\begin{aligned}
\forall Q.i\ \exists R\ S \vee (ctr \wedge B) \quad &\longrightarrow \quad \forall Q.i\ \exists R\ awake(S)\\
&\qquad\qquad \vee((ctr)^m \wedge awake(B))\\
&\qquad \text{where} \quad ctr \in \{X_i \in Forge_c(E,\mathscr{K}), X_i = u\}\\
&\qquad\qquad \text{and} \quad B \cap \mathscr{M}(\overrightarrow{X}) = \emptyset
\end{aligned}$$

The transfer label rule transfers a master label for a vector variable $\overrightarrow{X}$ from a forge constraint to an equality one.

$$\begin{aligned}
&\forall Q.i.j \exists R \quad S \vee ((X_i \in Forge_c(E,\mathscr{K}))^m \wedge X_j = u \wedge B)\\
&\longrightarrow\\
&\forall Q.i.j\ \exists R \quad awake(S) \vee (X_i \in Forge_c(E,\mathscr{K}) \wedge (X_j = u)^m\\
&\qquad\qquad\qquad\qquad \wedge awake(B))
\end{aligned}$$

Other rules of the same group G_1 format constraints in order to get preferably variables on their left hand-side, or replace indexed variables by non-indexed ones.

G_2 contains the *Forge* rules. Here, we have for example :

$$\begin{aligned}
t \in Forge(E,\mathscr{K}) \quad &\longrightarrow \quad t \in Forge_c(E,\mathscr{K}) \vee\\
&\qquad \bigvee_{w\in E} t \in Sub(w,E,\mathscr{E},\mathscr{K})
\end{aligned}$$

This generic rule illustrates the two possible ways for forging a term t: either by composing or by decomposing one of the knowledge. The other rules enumerate all possible ways a term can be composed by the Intruder: we have exactly one rule for decomposing each kind of operator in the signature $\mathscr{G}$. In particular, there is one rule for the *mpair* operator where *mpair* autonomy is used to justify the quantification.

G_3 contains the *Sub* rules. These rules are similar to the *Forge* ones, but they decompose Intruder knowledge. In this

group, there is a generic rule :

$$\begin{array}{lll} t \in Sub(u,E,\mathscr{E},\mathscr{K}) & \longrightarrow & (t=u) \text{ if } u \in \mathscr{K} \\ t \in Sub(u,E,\mathscr{E},\mathscr{K}) & \longrightarrow & (t \in Sub_d(u,E,\mathscr{E},\mathscr{K})) \\ & & \vee (t=u) \text{ otherwise} \end{array}$$

This rule follows precisely the intruder deduction rules: a term t is an accessible subterm of u iff $t = u$ or there exists a direct subterm u' of u, derivable from u, with t being an accessible subterm of u'. Therefore, there exists exactly one rule for decomposing each kind of operator in $\mathscr{G}$ (apart from variables and constants).

G_4 encodes unification algorithm for terms in our system, and thus, the resolution of equality constraints. These rules simply consist in testing recursively the compatibility of each top operator in each term. Therefore, the only equality constraints remaining after an iteration of these rules are those assigning a value to a variable, i.e. $X = u$ with $X \in \mathscr{X} \cup \mathscr{X}_{\mathscr{J}}$. The goal of G_5 and G_6 is to replace variables by managing interactions between constraints.

G_5 aims at replacing variables by their value, inside one block and independently of other blocks. Consequently, these rules only consider multiple occurrences of the same variable, with the same index in case of indexed variable. For instance, the interaction between two *Equality* constraints is managed by Rules 26 and 27 for respectively indexed and non indexed variables.

$$\begin{array}{lll} B \wedge (X_i = u)^{\bullet} \wedge (X_i = v)^{\bullet} & \longrightarrow & B \wedge (X_i = u)^{\bullet} \\ & & \wedge (X_i = v)^{\bullet} \wedge u = v \end{array} \quad (26)$$

$$(X = u)^{sm} \wedge (X = v) \longrightarrow (X = u)^{sm} \wedge (u = v) \quad (27)$$

Besides, interaction for *Forge* and *Sub* constraints is managed as follows:

$$A \in Forge_c(E,\mathscr{K}) \wedge t \in Sub_d(A,E',\mathscr{E},\mathscr{K}) \longrightarrow \bot \quad \text{where } A \in \mathscr{X} \cup \mathscr{X}_{\mathscr{J}}$$

This rule says that it is not necessary to decompose a variable. While a bit more complex than expected, our semantics of $Sub_d(,,,)$ has been defined to prevent useless actions like this, thus ensuring the validity of G_5.

G_6 generalizes G_5 by allowing variable replacements from one block to an other one. Given a constraint containing a variable X_i that must be replaced by its value, we enumerate a finite number of "candidate" terms representing all possible values of this variable according to the whole constraints system. These values are provided by master constraints. For instance, the interleaving between a *Sub* constraint with constraints of other blocks leads to the following rule:

$$t \in Sub_d(X_m,E',\mathscr{E},\mathscr{K}) \longrightarrow \bigvee_{(X_i=u)\in\mathscr{E}} \exists k'\ t \in Sub(u\delta,E',\mathscr{E},\mathscr{K}) \wedge (X_m = u\delta)^f \wedge X_m \notin Forge_c(E',\mathscr{K}) \text{ with } \delta = \delta_{i,m}^{k'}$$

In this rule, only the case where master constraints are *equality* ones are taken into account since the interleaving with *Forge* ones leads to $\bot$. This rule adds an (extra) equality representing the master constraint it used, but labeled final to prevent further reductions on it. It adds also negative constraints to eliminate the case of *Forge* master constraints.

Solved constraints. When none of our rules can be applied to our constraints system, solved constraints only contain elementary constraints of the following type: a constraint of type: $(X_i \in Forge_c(E,\mathscr{K}))^{\bullet}$, $X \in Forge_c(E,\mathscr{K})$, $(Y \notin Forge_c(E,\mathscr{K}))$, $(X_i = u)^{\bullet}$, $(X = u)^{sm}$, $(\forall j\ X_i \neq u)$ where $X \in \mathscr{X}$, $Y \in \mathscr{X} \cup \mathscr{X}_{\mathscr{J}}$, $X_i \in \mathscr{X}_{\mathscr{J}}$, $u \in \mathscr{T}$, $j \in Var_{\mathscr{J}}(u)$, $E \subset \mathscr{T}$ and $\mathscr{K} \subset \mathscr{T}$.

4. Verification of well-tagged protocols with autonomous keys

We introduce here the verification algorithm for the class of well-tagged protocols with autonomous keys. Given a set R of inference rules and a formula F we say that $R(F)$ is a *closure* of F by R if it is derived by a finite number of applications of rules in R and no rule can be further applied to $R(F)$. First of all, we define a reduction of equalities chain in an environment, given the set of index variables quantified universally Q:

Notation 2 ($\lceil\mathscr{E}\rceil_Q$): We note $\lceil\mathscr{E}\rceil_Q$ the closure of $\mathscr{E}$ by the following rules:

$$\begin{array}{lll} X = Y \wedge Y = u & \longrightarrow & X = u \wedge Y = u \\ X_i = Y_i \wedge Y_j = u & \longrightarrow & \exists k'\ X_i = u\delta_{Q,i}^{k'} \wedge Y_j = u \end{array}$$

for $X, Y \in \mathscr{X} \cup \mathscr{X}_{\mathscr{J}}$, $X_i, Y_j \in \mathscr{X}_{\mathscr{J}}$, $u \notin \mathscr{X} \cup \mathscr{X}_{\mathscr{J}}$ and k' fresh.

Second, we introduce the normalization function denoted by $S \mapsto (S)\downarrow$. A normalization of a constraints system can be defined when some closures can be computed as follows using a subset of the inference rules. This normalization operates in two main phases:

Definition 17: [Normalization function] Let S be a block system. We denote by *SR* the whole set of inference rules except the labelling rule. We assume that we can compute:

Phase 1:	S_1, a closure of S through *SR* except Rules 26 and 27;
Labelling:	S_2, a closure of S_1 through the labelling and label transfer rules;
Phase 2:	S_3, a closure of S_2 through *SR*. This closure is denoted by $(S)\downarrow$.

The *Labelling* step adds labels for creating master constraints, making sure to always favor labelling of equality constraints to a forge constraints. While Phases 1 and 2 are similar by the rules they use, their behaviors differ: when used in our algorithm for a step $R_i \Rightarrow S_i$, Phase 1 will never use any constraint interleaving rule with a master constraint $\mathscr{M}(\overrightarrow{X})$ for a variable that has just appeared in the current

step (i). This means that during Phase 1, the variables of the current step cannot be replaced yet from a block to an other, simply because none of them have master constraints yet. However, the second phase do not have this limitation. The verification algorithm is the following:

Algorithm 1: Let $P = \{R_i \Rightarrow S_i | i = 1..k\}$ be Well-Tagged, $Sec \in \mathscr{T}$, $R_{k+1} \triangleq Sec$ and $S_0 \subset \mathscr{T}_g$.

1) Let $CBS_0 \triangleq \forall Q \exists R \top$, with $Q = R = \emptyset$, be the initial constraints system.
2) For i from 1 to $k+1$:
   a) Assume that $CBS_{i-1} \triangleq \forall Q \exists R\ (B_1, \mathscr{E}_{i-1,1}) \vee .. \vee (B_p, \mathscr{E}_{i-1,p})$;
   b) Let $ctr_i \triangleq R_i \in Forge(S_0, S_1, .., S_{i-1}, \emptyset)$;
   c) Let $\mathscr{E}_i = \bigcup_{\overrightarrow{X}} \mathscr{M}(CBS_{i-1}, \overrightarrow{X})$ and for all $j = 1, ..p$, $X, Y \in \mathscr{X} \cup \mathscr{X}_{\mathscr{I}}$, $\mathscr{E}_{i,j} = \left[\mathscr{E}_i \cup \mathscr{SM}(B_j, \mathscr{X})\right] \setminus \{(X = Y)\}$;
   d) Let $CBS_i \triangleq (\forall Q \exists R\ (B_1 \wedge ctr_i, \mathscr{E}_{i,1}) \vee .. \vee (B_p \wedge ctr_i, \mathscr{E}_{i,p}))\downarrow$
3) Test Satisfiability of CBS_{k+1} (return insecure iff satisfiable).

Note that sets $\mathscr{E}_i, \mathscr{E}_{i,j}$ denotes respectively the set of master constraints for vector variables and the set of submaster constraints for variables of block B_j, both with variables already processed in previous steps (strictly before step i). Notation $\top$ represents true. The algorithm chooses a "possible" protocol run represented by π, and tests if after this run *Sec* is derivable by the intruder for some length e of $mpair(,)$. We test this by increasing the initial constraints system CBS_0 with each protocol step successively, and by normalizing the resulting constraints system at each step. This step-by-step normalization is required by our inference rules which assumes that master and sub-master constraints for previous steps have been already computed.

The application of this algorithm on our running example is given in Example 8.

Example 8: Suppose that the intruder has as knowledge E at Step 2 of the protocol, where $E = \{mpair(i, A \centerdot \{\!|s_i|\!\}_k), mpair(l, \{\!|f(r_l)|\!\}_k)\}$ with a hash function f. Terms with capital letters are variables. The first step of this protocol is represented by:

$$mpair(i, a_i \centerdot \{\!|S_i|\!\}_k) \in Forge(E, \emptyset)$$

Solving this step leads to the following major steps of normalization. First, we have:

$$\begin{array}{l} \forall i \forall j \\ ((A = a_i) \wedge (S_i = s_i)) \\ \vee ((A = mpair(i, a_i \centerdot \{\!|S_i|\!\}_k))) \\ \vee a_j \centerdot \{\!|S_j|\!\}_k \in Forge(E, \emptyset) \end{array}$$

The normalization of the third block leads to:

$$\begin{array}{l} \forall i \forall j \exists l \\ ((A = a_i) \wedge (S_i = s_i)) \\ \vee ((A = mpair(i, a_i \centerdot \{\!|S_i|\!\}_k))) \\ \vee ((A = a_j) \wedge (S_j = s_l)) \\ \vee ((a_j \in Forge(E, \emptyset)) \wedge (\{\!|S_j|\!\}_k \in Forge(E, \emptyset))) \end{array}$$

The normalization of the last block leads to:

$$\begin{array}{l} \forall i \forall j \exists l \exists m \exists o \\ ((A = a_i) \wedge (S_i = s_i)) \\ \vee ((A = mpair(i, a_i \centerdot \{\!|S_i|\!\}_k))) \\ \vee ((A = a_j) \wedge (S_j = s_l)) \\ \vee ((A = a_j) \wedge ((A = \{\!|S|\!\}_k) \wedge ((S = S_j)) \\ \vee ((A = a_j) \wedge (S_j = s_m)) \\ \vee ((A = a_j) \wedge (S_j = f(r_o))) \end{array}$$

Finally, labelling constraints has as result the following constraints system:

$$\begin{array}{l} \forall i \forall j \exists l \exists m \exists o \\ ((A = a_i)^{sm} \wedge (S_i = s_i)^m) \\ \vee ((A = mpair(i, a_i \centerdot \{\!|S_i|\!\}_k))^{sm}) \\ \vee ((A = a_j)^{sm} \wedge (S_j = s_l)^m) \\ \vee ((A = a_j)^{sm} \wedge (S_j = s_m)^m) \\ \vee ((A = a_j)^{sm} \wedge (S_j = f(r_o))^m) \end{array}$$

Simplification rules with tagging. We have defined our constraints simplification for untagged terms. Nevertheless, these rules deal also with tagged terms following the definition of our signature. Indeed, for decomposition in *Forge* or *Sub* constraints, our rules behave similarly with tagged terms as for untagged terms. Moreover, for equality constraint, the constraint $(X_i)^i = u$ leads to $\bot$ when u is untagged, since $(X_i)^i = [e_i, X_i]$. Besides, for replacement in *Forge* or *Sub* constraints, tagged variables behave as 'special' variable. For example, given a constraint $(t \in Sub(X_i^i, E, \mathscr{E}, \mathscr{K}))$, we search for master constraints for the vector $\overrightarrow{X}$. Then, assuming $(X_o = u)^m$ is one of them, the replacement result would $(t \in Sub((u\delta)^i, E, \mathscr{E}, \mathscr{K}))$ for this master constraint. The same reasoning is valid for a *Forge* constraint.

Control for Termination of Normalization. To prevent cycles, we have added some control on rule application. This will be crucial to get the termination proof. We first extend the blocks of constraints to include an history, that is, a sorted list of rules that have applied at some step to an ancestor of this block. Any rule application will implicitly update the history of the block to which it is applied. This history is used in some rules to prevent the generation of a constraint that has already appeared in the history and to prevent the generation of fresh indexes (generated with the deduced constraint).

We also introduce a notion of *anteriority* between index variables from the same block The motivation here is to guarantee that each time we have a replacement of index variables in the block, we also preserve either the universally quantified ones or the anterior one (in case the two indexes are existentially quantified). This will be used to bound the number of existential indexes generated in the normalization

of constraints since in case of replacement of an index variable, we either use an universal index for which we already know a bound or we re-use an anterior index which is already computed in the set of existential indexes.

Finally, we define the dependency graph of a block that is used to detect variable occurrence cycles since these cycles entail unsatisfiable constraints of the form $X = u$ where X is a variable occurring in u.

These controls allow us to obtain the following proposition :

Proposition 1: (Termination of normalization) Our Verification Algorithm [6] terminates for Well-Tagged protocols with Autonomous Keys.

Sketch of proof: The proof of this theorem is done in two steps:

First Step: Bounding the number of possible indexes to be generated. We first bound the number of *universal* indexes generated. Then, we prove that the set of *existential* indexes is finite. This is proved incrementally on subsets of constraints.

Second Step: Defining a weight for a constraints system that decreases for each rule application. The weight of a term in a block is defined as the sum of weights of its symbols. Hash and mpair operators have weight 2. Other functions have weight 1. The weight of a variable X is defined such that the weight of the block containing X decreases when we replace X by its value in this block (for example replacing $X = u$ by $v = u$ when $X = v$ belongs to the block).

The weight of an elementary constraint is defined as a lexicographic order on first the number of keys not in $\mathscr{K}$ and second a numerical measure over the constraint parameters. This measure is defined such that the weight of *Forge* constraints is higher than the weight of $Forge_c$ constraints or *Sub* ones. In the same way, the weight of *Sub* constraints is higher than the weight of Sub_d constraints and equality ones.

Since thanks to the first step, we have a bound on the number of indexes generated, we know the bound on the number of Forge($X \in Forge_c(E, \mathscr{K})$), equality ($X = u$), or negative constraints where $X \in \mathscr{X} \cup \mathscr{X}_{\mathscr{I}}$. The weight of a blocks system is defined as a lexicographic order that *decreases* when:

1) the number of replacement of indexed variables in the block history increases,
2) a *final* constraint appears in the block,
3) a *negative* constraint appears in the block,
4) a *submaster* constraint is generated in the block,
5) a *master* constraint of type *equality* is generated in the block
6) a *master* constraint of type *Forge* is generated in the block
7) the *multiset* of the weights of the constraints of the block decreases.
8) a constraint of the form $X = u$ where $X \in \mathscr{X} \cup \mathscr{X}_{\mathscr{I}}$ is generated in the block.

Note that the list of these facts is given in the lexicographic order. For example, the weight of the blocks system decreases if a negative constraint appears in the block *and* the number of final constraints remains the same.

The weight of a constraints system is defined as the multiset of the weights of its different blocks. For full details, see Technical Report [5]. □

Besides we have shown that the normalization preserves the semantics, $[\![CBS_{k+1}]\!]^e_\emptyset \neq \emptyset$, iff the protocol run defined by π has an attack. Thus, assuming the next proposition, the correctness and completeness of the algorithm follows :

Proposition 2: (Correctness and Completeness of Normalization) Let CBS_i and ctr_i ($i = 1..k+1$) be as in the verification algorithm, for some *Well-tagged* protocol P with autonomous keys. Then for all e,

$$[\![CBS_{i-1} \wedge ctr_i]\!]^e_\emptyset = \left|\left[\ (CBS_{i-1} \wedge ctr_i)\downarrow\ \right]\right|^e_\emptyset$$

Finally, once the constraints system is normalized, it can be checked for satisfiability and then knowing if it exists a solution or not, i.e. if it exists an attack or the protocol is secure. This result is given in Proposition 3.

Proposition 3: (Satisfiability of normalized form) When Algorithm 1 is applied to a Well-Tagged protocol P with autonomous keys, the satisfiability of the resulting normalized constraints system can be decided.

Theorem 2: The insecurity problem for Well-Tagged protocols with Autonomous Keys is decidable.

Proof: The proof of Theorem 2 is a direct consequence of Propositions 1, 2 and 3. The whole proofs of these propositions can be found in the technical report [5]. □

Moreover, it is worth noticing that the restriction of autonomous keys is only used for proofs of termination. Thus, our inference rules are correct and complete. The normalization of a constraints system lead to a normalized constraints system whose satisfiability can be checked.

5. Conclusion

We have proposed an extension of the constraint-based approach in symbolic protocol verification in order to handle a class of protocols (the Well-Tagged ones with Autonomous Keys) which admit unbounded lists in messages. This class can be used in particular to model interesting group protocols. We have proved termination of normalization for this class of protocols. The combination with a previous work [6] allows us to prove that the insecurity problem for Well-Tagged protocols with Autonomous Keys is decidable.

We have studied in a previous work the Asokan-Ginzboorg protocol with keys having index variables outside

mpair's. We showed that the analysis of this protocol terminates. Thus, we conjecture termination for a larger class than Well-Tagged protocols where the restriction of autonomous keys is weakened to include at least this protocol.

References

[1] Armando, A., Compagna, L.: Automatic SAT-Compilation of Protocol Insecurity Problems via Reduction to Planning. In: Foundation of Computer Security & Verification Workshops. Copenhagen, Denmark (2002)

[2] Asokan, N., Ginzboorg, P.: Key agreement in ad hoc networks. Computer Communications **23**(17), 1627–1637 (2000)

[3] Basin, D., Mödersheim, S., Viganò, L.: An On-The-Fly Model-Checker for Security Protocol Analysis. In: E. Snekkenes, D. Gollmann (eds.) Proceedings of ESORICS'03, LNCS 2808, pp. 253–270. Springer-Verlag (2003)

[4] Blanchet, B., Podelski, A.: Verification of cryptographic protocols: Tagging enforces termination. In: A.D. Gordon (ed.) 6th International Conference, FOSSACS 2003, Warsaw, Poland, April 7-11, 2003, Proceedings, *Lecture Notes in Computer Science*, vol. 2620, pp. 136–152. Springer (2003)

[5] Chridi, N., Turuani, M., Rusinowitch, M.: Constraints-based Verification of Parameterized Cryptographic Protocols. Research Report RR-6712, INRIA (2008). URL http://hal.inria.fr/inria-00336539/en/

[6] Chridi, N., Turuani, M., Rusinowitch, M.: Towards a Constrained-based Verification of Parameterized Cryptographic Protocols. In: M. Hanus (ed.) LOPSTR 2008: Logic-based Program Synthesis and Transformation. Valencia, Spain (2008). URL http://hal.inria.fr/inria-00332484/en/

[7] Chridi, N., Vigneron, L.: Strategy for Flaws Detection based on a Services-driven Model for Group Protocols, chap. 24, pp. 361–370. Future and Trends in Constraint Programming. ISTE (2007)

[8] Dolev, D., Yao, A.: On the security of public key protocols. IEEE Trans. Inform Theory IT-29 pp. 198–208 (1983). Also STAN-CS-81-854, May 1981, Stanford U.

[9] J.A.Bull, D.J.Otway: The authentication protocol. Tech. rep., Defence Research Agency, Mavern,UK (1997)

[10] K.O.Kürtz, R.Küsters, T.Wilke: Selecting theories and nonce generation for recursive protocols. In: FMSE '07: Proceedings of the 2007 ACM workshop on Formal methods in security engineering, pp. 61–70. ACM, New York, NY, USA (2007)

[11] Kremer, S., Mercier, A., Treinen, R.: Proving group protocols secure against eavesdroppers. In: IJCAR 08, Lecture Notes in Artificial Intelligence, pp. 116–131. Springer-Verlag, Berlin, Heidelberg (2008)

[12] Küsters, R., Truderung, T.: On the automatic analysis of recursive security protocols with xor. Tech. rep., ETH Zurich (2007). An abridged version appears in STACS 2007

[13] Küsters, R., Wilke, T.: Automata-based analysis of recursive cryptographic protocols. In: 21st Symposium on Theoretical Aspects of Computer Science (STACS 2004), Lecture Notes in Computer Science. Springer-Verlag (2004)

[14] Martinelli, F.: Analysis of security protocols as open systems. Theor. Comput. Sci. **290**(1), 1057–1106 (2003)

[15] Meadows, C., Syverson, P., Cervesato, I.: Formalizing gdoi group key management requirements in npatrl. In: In Proceedings of the ACM Conference on Computer and Communications Security. ACM, pp. 235–244. ACM Press (2001)

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

[17] Mittra, S.: Iolus: A framework for scalable secure multicasting. In: SIGCOMM, pp. 277–288 (1997)

[18] Molva, R., Michiardi, P.: Security in ad hoc networks. In: PWC, pp. 756–775 (2003)

[19] Paulson, L.C.: Mechanized proofs for a recursive authentication protocol. In: 10th Computer Security Foundations Workshop, pp. 84–95. IEEE Computer Society Press (1997)

[20] Pereira, O., Quisquater, J.J.: Some attacks upon authenticated group key agreement protocols. Journal of Computer Security **11**(4), 555–580 (2003)

[21] Ramanujam, R., Suresh, S.P.: Tagging makes secrecy decidable with unbounded nonces as well. In: P.K. Pandya, J. Radhakrishnan (eds.) FST TCS 2003: Foundations of Software Technology and Theoretical Computer Science, 23rd Conference, Mumbai, India, December 15-17, 2003, Proceedings, *Lecture Notes in Computer Science*, vol. 2914, pp. 363–374. Springer (2003)

[22] Rusinowitch, M., Turuani, M.: Protocol insecurity with a finite number of sessions, composed keys is np-complete. Theor. Comput. Sci. **1-3**(299), 451–475 (2003)

[23] Steel, G., Bundy, A.: Attacking group multicast key management protocols using CORAL. In: A. Armando, L. Viganó (eds.) Proceedings of the ARSPA Workshop, *ENTCS*, vol. 125, pp. 125–144 (2004)

[24] Steiner, M., Waidner, M., Tsudik, G.: Cliques: A new approach to group key agreement. In: ICDCS '98: Proceedings of the The 18th International Conference on Distributed Computing Systems, p. 380. IEEE Computer Society, Washington, DC, USA (1998)

[25] Taghdiri, M., Jackson, D.: A lightweight formal analysis of a multicast key management scheme. In: FORTE, pp. 240–256 (2003)

[26] Truderung, T.: Selecting theories and recursive protocols. CONCUR 2005 - Concurrency Theory pp. 217–232 (2005)

[27] Weidenbach, C.: Towards an automatic analysis of security protocols in first-order logic. In: 16th International Conference on Automated Deduction, *LNCS*, vol. 1632, pp. 314–328. Springer (1999)