Handling Algebraic Properties in Automatic Analysis of Security Protocols*

Y. Boichut, P.-C. Héam and O. Kouchnarenko

LIFC, FRE 2661 CNRS, Besançon, France, INRIA/CASSIS
{boichut,heampc,kouchna}@lifc.univ-fcomte.fr

Abstract. This paper extends the approximation-based theoretical framework in which the security problem – secrecy preservation against an intruder – may be semi-decided through a reachability verification.
We explain how to cope with algebraic properties for an automatic approximation-based analysis of security protocols. We prove that if the initial knowledge of the intruder is a regular tree language, then the security problem may by semi-decided for protocols using cryptographic primitives with algebraic properties. More precisely, an automatically generated approximation function enables us 1) an automatic normalization of transitions, and 2) an automatic completion procedure. The main advantage of our approach is that the approximation function makes it possible to verify security protocols with an arbitrary number of sessions.
The concepts are illustrated on an example of the *view-only* protocol using a cryptographic primitive with the exclusive or algebraic property.
Keywords: Security protocol, algebraic properties, automatic verification, approximation

1 Introduction

Cryptographic protocols are widely used to secure the exchange of information over open modern networks. It is now widely accepted that formal analysis can provide the level of assurance required by both the developers and the users of the protocols. However, whatever formal model one uses, analyzing cryptographic protocols is a complex task because the set of configurations to consider is very large, and can even be infinite. Indeed, any number of sessions (sequential or parallel executions) of protocols, sessions interleaving, any size of messages, algebraic properties of encryption or data structures give rise to infinite-state systems.

Our main objective is to automate in so far as possible the analysis of protocols. More precisely, we are interested in a fully automatic method to (semi)decide the *security* problem. In the context of the verification of protocols, the security problem consists of deciding whether a protocol preserves secrecy against an intruder, or not.

For this problem, current verification methods based on model-checking can be applied whenever the number of participants and the number of sessions between the agents are bounded. In this case, the protocol security problem is co-NP-complete [21]. The recent work [22] presents new decidability results for a bounded

* This work has been supported by the European project AVISPA IST-2001-39252 and the French projects RNTL PROUVE and ACI SATIN.

number of sessions, when the initial knowledge of the intruder is a regular language under the assumption that the keys used in protocols are atomic.

When the number of sessions is unbounded, the security problem of cryptographic protocols becomes undecidable, even when the length of the messages is bounded [15]. Decidability can be recovered by adding restrictions as in [13] where tree automata with one memory are applied to decide secrecy for cryptographic protocols with single blind copying.

Another way to circumvent the problem is to employ abstraction-based approximation methods [19,17]. In fact, these methods use regular tree languages to approximate the set of messages that the intruder might have acquired during an unbounded number of sessions of protocols. In this framework, the security problem may be semi-decided through a reachability verification. The finite tree automata permit to ensure that some states are unreachable, and hence that the intruder will never be able to know certain terms.

To achieve the goal of an automatic analysis of protocols, we have investigated, improved [7] and extended [6] the semi-algorithmic method by Genet and Klay. The main advantage of our approach is that the automatically generated symbolic approximation function makes it possible to automatically verify security protocols while considering an unbounded number of sessions. The efficiency and usefulness of our approach have been confirmed by the tool `TA4SP` (Tree Automata based on Automatic Approximations for the Analysis of Security Protocols), which has already been used for analyzing many real Internet security protocols as exemplified in the European Union project AVISPA [1] [2].

However, for some cryptographic protocols, the secrecy is not preserved even when used with strong encryption algorithms. The purpose of the work we present in this paper is to extend the symbolic approximation-based theoretical framework defined in [6] to security protocols using cryptographic primitives with algebraic properties. To be more precise, the goal is to relax the perfect cryptography assumption in our symbolic approximation-based method. As explained in [14], such an assumption is too strong in general since some attacks are built using the interaction between protocol rules and properties of cryptographic operators.

The main contribution of this paper consists of showing the feasibility of the automatic analysis of protocols where the number of sessions is unbounded and some algebraic properties of the cryptographic primitives – e.g. the exclusive or – are taken into account.

The main result of the paper is that the automatically generated approximation function allows us to over-approximate the knowledge of the intruder. `TA4SP` we have been implementing is used for obtaining experimental results. The most important new feature is the exclusive or algebraic property, `XOR` for short, modulo which the protocol analysis is performed. The feasibility of our approach is illustrated on the example of the *view-only* protocol.

[1] http://www.avispa-project.org/

Related Work In [20] it has been shown that using equational tree automata under associativity and/or commutativity is relevant for security problems of cryptographic protocols with an equational property. For protocols modeled by associative-commutative TRSs, the authors announce the possibility for the analysis to be done automatically thanks to the tool ACTAS manipulating associative-commutative tree automata and using approximation algorithms. However, the engine has still room to be modified and optimized to support an automated verification.

In [23], the authors investigate algebraic properties and timestamps in the approximation-based protocol analysis. Like in [6], there is no left-linearity condition on TRSs modeling protocols. However, the weakness of the work is that no tool is mentioned in [23].

In the recent survey [14], the authors give an overview of the existing methods in formal approaches to analyze cryptographic protocols. In the same work, a list of some relevant algebraic properties of cryptographic operators is established, and for each of them, the authors provide examples of protocols or attacks using these properties.

This survey lists two drawbacks with the recent results aiming at the analysis of protocols with algebraic properties. First, in most of the papers a particular decision procedure is proposed for a particular property. Second, the authors emphasize the fact that the results remain theoretical, and very few implementations automatically verify protocols with algebraic properties.

Following the result presented in [10], the authors have prototyped a new feature to handle the `XOR` operator in CL-AtSe (Constraint Logic based Attack Searcher), one of the four official tools of the `AVISPA` tool-set [2].

Layout of the paper The paper is organized as follows. After giving preliminary notions on tree-automata and term rewriting systems (TRSs), we introduce in Section 2 a substitution notion depending on rules of a TRS, and a notion of compatibility between that substitutions and finite tree-automata, both suitable for our work. Section 3 presents the completion theorem making the completion procedure stop even when protocols – modeled by non left-linear TRSs – use cryptographic primitives with algebraic properties. The main result is then a consequence of the completion theorem allowing us to use an approximation function to obtain an over-approximation of the knowledge of the intruder. In Section 4, we explain how to apply the main theorem to verify the *view-only* protocol using a cryptographic primitive with the exclusive or algebraic property.

2 Background and notation

In this section basic notions on finite tree automata, term rewriting systems and approximations are reminded. The reader is refereed to [12] for more detail.

2.1 Notations

For any set A, we denote by 2^A the set of subsets of A. We denote by $\mathbb{N}$ the set of natural integers and $\mathbb{N}^*$ denotes the finite strings over $\mathbb{N}$.

Let $\mathcal{F}$ be a finite set of symbols with their arities. The set of symbols of $\mathcal{F}$ of arity i is denoted $\mathcal{F}_i$. Let $\mathcal{X}$ be a finite set whose elements are variables. We assume that $\mathcal{X} \cap \mathcal{F} = \emptyset$.

A finite ordered tree t over a set of labels $(\mathcal{F}, \mathcal{X})$ is a function from a prefix-closed set $\mathcal{P}os(t) \subseteq \mathbb{N}^*$ to $\mathcal{F} \cup \mathcal{X}$. A term t over $\mathcal{F} \cup \mathcal{X}$ is a labeled tree whose domain $\mathcal{P}os(t)$ satisfies the following properties:

- $\mathcal{P}os(t)$ is non-empty and prefix closed,
- For each $p \in \mathcal{P}os(t)$, if $t(p) \in \mathcal{F}_n$, then $\{i \mid p.i \in \mathcal{P}os(t)\} = \{1, \ldots, n\}$,
- For each $p \in \mathcal{P}os(t)$, if $t(p) \in \mathcal{X}$, then $\{i \mid p.i \in \mathcal{P}os(t)\} = \emptyset$.

Each element of $\mathcal{P}os(t)$ is called a position of t. For each subset $\mathcal{K}$ of $\mathcal{X} \cup \mathcal{F}$ and each term t we denote by $\mathcal{P}os_{\mathcal{K}}(t)$ the subset of positions p's of t such that $t(p) \in \mathcal{K}$. Each position p of t such that $t(p) \in \mathcal{F}$, is called a functional position. The set of terms over $(\mathcal{F}, \mathcal{X})$ is denoted $\mathcal{T}(\mathcal{F}, \mathcal{X})$. A ground term is a term t such that $\mathcal{P}os(t) = \mathcal{P}os_{\mathcal{F}}(t)$ (i.e. such that $\mathcal{P}os_{\mathcal{X}}(t) = \emptyset$). The set of ground terms is denoted $\mathcal{T}(\mathcal{F})$.

A subterm $t_{|p}$ of $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ at position p is defined by the following:

- $\mathcal{P}os(t_{|p}) = \{i \mid p.i \in \mathcal{P}os(t)\}$,
- For all $j \in \mathcal{P}os(t_{|p})$, $t_{|p}(j) = t(p.j)$.

We denote by $t[s]_p$ the term obtained by replacing in t the subterm $t_{|p}$ by s.

If $\mathcal{X}$ contains n elements and is (arbitrarily) ordered, then a context C is an element of $\mathcal{T}(\mathcal{F}, \mathcal{X})$ in which each element of $\mathcal{X}$ occurs exactly once. The expression $C[t_1, \ldots, t_n]$ for $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ denotes the term in $\mathcal{T}(\mathcal{F}, \mathcal{X})$ obtained from C by replacing the i-th element of $\mathcal{X}$ by t_i for each $1 \leq i \leq n$.

For all sets A and B, we denote by $\Sigma(A, B)$ the set of functions from A to B. If $\sigma \in \Sigma(\mathcal{X}, B)$, then for each term $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, we denote by $t\sigma$ the term obtained from t by replacing for each $x \in \mathcal{X}$, the variable x by $\sigma(x)$.

A term rewriting system (TRS for short) over $\mathcal{T}(\mathcal{F}, \mathcal{X})$ is a finite set of pairs (l, r) from $\mathcal{T}(\mathcal{F}, \mathcal{X}) \times \mathcal{T}(\mathcal{F}, \mathcal{X})$, denoted $l \to r$, such that the set of variables occurring in r is included in the set of variables of l. A term rewriting system is left-linear if for each rule $l \to r$, every variable occurring in l occurs at most once. For each ground term t, we denote by $\mathcal{R}(t)$ the set of ground terms t' such that there exist a rule $l \to r$ of $\mathcal{R}$, a function $\mu \in \Sigma(\mathcal{X}, \mathcal{T}(\mathcal{F}))$ and a position p of t satisfying $t_{|}p = l\mu$ and $t' = t[r\mu]_p$. The relation $\{(t, t') \mid t' \in \mathcal{R}(t)\}$ is classically denoted $\to_{\mathcal{R}}$. For each set of ground terms B we denote by $\mathcal{R}^*(B)$ the set of ground terms related to an element of B modulo the reflexive-transitive closure of $\to_{\mathcal{R}}$.

A tree automaton $\mathcal{A}$ is a tuple $(\mathcal{Q}, \Delta, F)$, where $\mathcal{Q}$ is the set of states, Δ the set of transitions, and F the set of final states. Transitions are rewriting rules of the

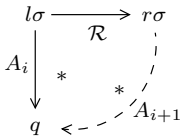
form $f(q_1, \ldots, q_k) \to q$, where $f \in \mathcal{F}_k$ and the q_i's are in $\mathcal{Q}$. A term $t \in \mathcal{T}(\mathcal{F})$ is accepted or recognized by $\mathcal{A}$ if there exists $q \in F$ such that $t \to_{\Delta}^{*} q$ (we also write $t \to_{\mathcal{A}}^{*} q$). The set of terms accepted by $\mathcal{A}$ is denoted $\mathcal{L}(\mathcal{A})$. For each state $q \in \mathcal{Q}$, we write $\mathcal{L}(\mathcal{A}, q)$ for the tree language $\mathcal{L}((\mathcal{Q}, \Delta, \{q\}))$. A tree automaton is finite if its set of transitions is finite.

2.2 Approximations to Handle Algebraic Properties

This section recalls the approximation-based framework we have been developing, and explains our objectives from a formal point of view.

Given a tree automaton $\mathcal{A}$ and a TRS $\mathcal{R}$ (for several classes of automata and TRSs), the tree automata completion [17,16] algorithm computes a tree automaton $\mathcal{A}_k$ such that $\mathcal{L}(\mathcal{A}_k) = \mathcal{R}^*(\mathcal{L}(\mathcal{A}))$ when it is possible (for the classes of TRSs covered by this algorithm see [16]), and such that $\mathcal{L}(\mathcal{A}_k) \supseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A}))$ otherwise.

The tree automata completion works as follows. From $\mathcal{A} = \mathcal{A}_0$ completion builds a sequence $\mathcal{A}_0, \mathcal{A}_1, \ldots, \mathcal{A}_k$ of automata such that if $s \in \mathcal{L}(\mathcal{A}_i)$ and $s \to_{\mathcal{R}} t$ then $t \in \mathcal{L}(\mathcal{A}_{i+1})$. If we find a fixpoint automaton $\mathcal{A}_k$ such that $\mathcal{R}^*(\mathcal{L}(\mathcal{A}_k)) = \mathcal{L}(\mathcal{A}_k)$, then we have $\mathcal{L}(\mathcal{A}_k) = \mathcal{R}^*(\mathcal{L}(\mathcal{A}_0))$ (or $\mathcal{L}(\mathcal{A}_k) \supseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A}))$ if $\mathcal{R}$ is not in one class of [16]). To build $\mathcal{A}_{i+1}$ from $\mathcal{A}_i$, we achieve a *completion step* which consists of finding *critical pairs* between $\to_{\mathcal{R}}$ and $\to_{\mathcal{A}_i}$. For a substitution $\sigma : \mathcal{X} \mapsto \mathcal{Q}$ and a rule $l \to r \in \mathcal{R}$, a critical pair is an instance $l\sigma$ of l such that there exists $q \in \mathcal{Q}$ satisfying $l\sigma \to_{\mathcal{A}_i}^{*} q$ and $r\sigma \not\to_{\mathcal{A}_i}^{*} q$. For every critical pair $l\sigma \to_{\mathcal{A}_i}^{*} q$ and $r\sigma \not\to_{\mathcal{A}_i}^{*} q$ detected between $\mathcal{R}$ and $\mathcal{A}_i$, $\mathcal{A}_{i+1}$ is constructed by adding new transitions to $\mathcal{A}_i$ such that it recognizes $r\sigma$ in q, i.e. $r\sigma \to_{\mathcal{A}_{i+1}} q$.



However, the transition $r\sigma \to q$ is not necessarily a normalized transition of the form $f(q_1, \ldots, q_n) \to q'$ and so has to be normalized first. For example, to normalize a transition of the form $f(g(a), h(q')) \to q$, we need to find some states q_1, q_2, q_3 and replace the previous transition by a set of normalized transitions: $\{a \to q_1, g(q_1) \to q_2, h(q') \to q_3, f(q_2, q_3) \to q\}$.

Assume that q_1, q_2, q_3 are new states, then adding the transition itself or its normalized form does not make any difference. Now, assume that $q_1 = q_2$, the normalized form becomes $\{a \to q_1, g(q_1) \to q_1, h(q') \to q_3, f(q_1, q_3) \to q\}$. This set of normalized transitions represents the regular set of non normalized transitions of the form $f(g^{\star}(a), h(q')) \to q$; which contains the transition initially we wanted to add amongst many others. Hence, this is an over-approximation. We could have made an even more drastic approximation by identifying q_1, q_2, q_3 with q, for instance.

The above method does not work for all TRSs. For instance, consider the tree automaton $\mathcal{A} = (\{q_1, q_2, q_f\}, \{A \to q_1, A \to q_2, f(q_1, q_2) \to q_f\}, \{q_f\})$ and the TRS

$\mathcal{R} = \{f(x,x) \rightarrow g(x)\}$. There is no substitution σ such that $l\sigma \rightarrow_{\mathcal{A}}^{*} q$, for a q in $\{q_1, q_2, q_f\}$. Thus, following the procedure, there is no transition to add. But $f(A,A) \in \mathcal{L}(\mathcal{A})$. Thus $g(A) \in \mathcal{R}(L(\mathcal{A}))$. Since $g(A) \notin \mathcal{L}(\mathcal{A})$, the procedure stops (in fact does not begin) before providing an over-approximation of $\mathcal{R}^*(\mathcal{L}(\mathcal{A}))$.

The TRSs used in the security protocol context are often non left-linear. Indeed, there are a lot of protocols that cannot be modeled by left-linear TRSs. Unfortunately, to be sound, the approximation-based analysis described in [17] requires the use of left-linear TRSs. Nevertheless, this method can still be applied to some non left-linear TRSs, which satisfy some weaker conditions. In [16] the authors propose new linearity conditions. However, these new conditions are not well-adapted to be automatically checked.

In our previous work [6] we explain how to define a criterion on $\mathcal{R}$ and $\mathcal{A}$ to make the procedure automatically work for industrial protocols analysis. This criterion ensures the soundness of the method described in [17,16].

However, to handle protocols the approach in [6] is based on a kind of constant typing. In this paper we go further and propose a procedure supporting a fully automatic analysis and handling – without typing – algebraic properties like `XOR` presented in Fig. 1.
Let us remark first that the criterion defined in [16] does not allow managing the above rule IV. Second, in [6] we have to restrict `XOR` operations to typed terms to deal with the rule IV.

However, some protocols are known to be flawed by type confusing attacks [14,8,11]. In order to cope with these protocols, a new kind of substitution is defined in Section 2.3, and a new left-linear like criterion is introduced in Section 3.

Notice that following and extending [6], our approach can be applied for any kinds of TRSs. Moreover, it can cope with exponentiation algebraic properties and this way analyse Diffie-Hellman based protocols.

```
xor(x,y) ⟶ xor(y,x)                    I.
xor(xor(x,y),z) ⟶xor(x,xor(y,z))       II.
xor(x,0) ⟶ x                           III.
xor(x,x) ⟶0                            IV.
```

Fig. 1. `XOR` properties

2.3 $(l \rightarrow r)$-substitutions

In this technical subsection, we define the notion of a $(l \rightarrow r)$-substitution suitable for the present work.

Definition 1. *Let $\mathcal{R}$ be a term rewriting system and $l \rightarrow r \in \mathcal{R}$. A $(l \rightarrow r)$-substitution is an application from $\mathcal{P}\mathrm{os}_{\mathcal{X}}(l)$ into $\mathcal{Q}$.*

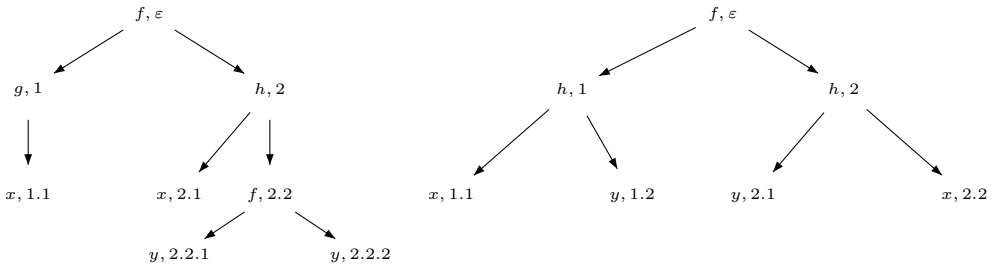
Let $l \to r \in \mathcal{R}$ and σ be a $(l \to r)$-substitution. We denote by $l\sigma$ the term of $\mathcal{T}(\mathcal{F}, \mathcal{Q})$ defined as follows:

- $\mathcal{P}\text{os}(l\sigma) = \mathcal{P}\text{os}(l)$,
- for each $p \in \mathcal{P}\text{os}(l)$, if $p \in \mathcal{P}\text{os}_{\mathcal{X}}(l)$ then $l\sigma(p) = \sigma(l(p))$, otherwise $l\sigma(p) = l(p)$.

Similarly, we denote by $r\sigma$ the term of $\mathcal{T}(\mathcal{F}, \mathcal{Q})$ defined by:

- $\mathcal{P}\text{os}(r\sigma) = \mathcal{P}\text{os}(r)$,
- for each $p \in \mathcal{P}\text{os}(r)$, if $p \notin \mathcal{P}\text{os}_{\mathcal{X}}(r)$ then $r\sigma(p) = r(p)$ and $r\sigma(p) = \sigma(l(p'))$ otherwise, where $p' = \min \mathcal{P}\text{os}_{r(p)}(l)$ (positions are lexicographically ordered).

Example 1 *Let us consider* $l = f(g(x), h(x, f(y, y)))$ *and* $r = f(h(x, y), h(y, x))$ *represented by the following trees (elements after the comma are the positions in the term;* l *is represented on the left and* r *on the right):*



Functional positions of l *are* 1.1 *and* 2.1 *for* x*, and* 2.2.1 *and* 2.2.2 *for* y*. Let* $\sigma(1.1) = q_1$*,* $\sigma(2.1) = q_2$*,* $\sigma(2.2.1) = q_3$ *and* $\sigma(2.2.2) = q_4$*;* σ *is a* $(l \to r)$*-substitution and*

$$l\sigma = f(g(q_1), h(q_2, f(q_3, q_4)))$$

is the term obtained from l *by substituting the variable in position* p *by* $\sigma(p)$*. Now we explain how to compute* $r\sigma$*. The minimal position where* x *[resp.* y*] occurs in* l *is* 1.1 *[resp.* 2.2.1*]. Thus* $r\sigma$ *is obtained from* r *by substituting all* x*'s in* r *by* $\sigma(1.1) = q_1$ *and all* y*'s by* $\sigma(2.2.1) = q_3$*. Thus*

$$r\sigma = f(h(q_1, q_3), h(q_3, q_1)).$$

As mentioned in Section 2.2, the completion procedure does not work for all tree automata and TRSs. That is why we introduce the notion of compatibility between finite tree-automata and $(l \to r)$-substitutions. Notice that the condition required below is weaker than the conditions in [16]. Moreover, it is more general and can be applied to a larger class of applications.

Definition 2. *Let* $\mathcal{A}$ *be a finite tree automaton. We say that a* $(l \to r)$*-substitution* σ *is* $\mathcal{A}$*-compatible if for each* $x \in \mathcal{V}\text{ar}(l)$*,*

$$\bigcap_{p \in \mathcal{P}\text{os}_{\{x\}}(l)} \mathcal{L}(\mathcal{A}, \sigma(p)) \neq \emptyset.$$

Example 2 *Let* $\mathcal{A}_{\text{exe}} = (\{q_0, q_f\}, \Delta_{\text{exe}}, \{q_f\})$ *with the set of transitions* $\Delta_{\text{exe}} = \{A \to q_0, A \to q_f, f(q_f, q_0) \to q_f, h(q_0, q_0) \to q_0\}$. *Let* $\mathcal{R}_{\text{exe}} = \{f(x, h(x, y)) \to h(A, x)\}$. *The automaton* $\mathcal{A}_{\text{exe}}$ *recognizes the set of trees such that every path from the root to a leaf is of the form* f^*h^*A. *Let us consider the substitution* σ_{exe} *defined by* $\sigma_{\text{exe}}(1) = q_f$, $\sigma_{\text{exe}}(2.1) = q_0$ *and* $\sigma_{\text{exe}}(2.2) = q_0$. *The tree* $t = A \to q_f$ *belongs to* $\mathcal{L}(\mathcal{A}, \sigma_{\text{exe}}(1))$. *Furthermore* $t = A \to q_0$, *so* $t \in \mathcal{L}(\mathcal{A}, \sigma_{\text{exe}}(2.2))$. *Therefore* σ_{exe} *is* $\mathcal{A}$*-compatible.*

3 Approximations for non-left Linear TRSs

In this section $\mathcal{R}$ denotes a fixed term rewriting system and $\mathcal{Q}$ an infinite set of states. We first introduce the notion of normalization associated with $(l \to r)$-substitutions. Secondly, we give the main result – consequence of the completion theorem – allowing us to over-approximate the descendants of regular sets.

3.1 Normalizations

The notion of normalization is common. The definitions below are simply adapted to our notion of $(l \to r)$-substitutions.

Definition 3. *Let* $\mathcal{A}$ *be a finite tree automaton. An approximation function (for* $\mathcal{A}$*) is a function which associates to each tuple* $(l \to r, \sigma, q)$, *where* $l \to r \in \mathcal{R}$, σ *is an* $\mathcal{A}$*-compatible* $(l \to r)$*-substitution and* q *a state of* $\mathcal{A}$, *a function from* $\mathcal{P}\text{os}(r)$ *to* $\mathcal{Q}$.

Example 3 *Consider the automaton* $\mathcal{A}_{\text{exe}}$, *the term rewriting system* $\mathcal{R}_{\text{exe}}$ *and the substitution* σ_{exe} *defined in Example 2. For* σ_{exe}, *an approximation function* γ_{exe} *may be defined by*

$$\gamma_{\text{exe}}(l \to r, \sigma_{\text{exe}}, q_0) : \{\varepsilon \mapsto q_1,\ 1 \mapsto q_2,\ 2 \mapsto q_f\}$$

To totally define γ_{exe}, *the others (finitely many)* $\mathcal{A}_{\text{exe}}$*-compatible substitutions should be considered too.*

The notion of normalization below is classical. The definition takes our notion of $(l \to r)$-substitutions into account only.

Definition 4. *Let* $\mathcal{A} = (\mathcal{Q}_0, \Delta, F_0)$ *be a finite tree automaton,* γ *an approximation function for* $\mathcal{A}$, $l \to r \in \mathcal{R}$, σ *an* $\mathcal{A}$*-compatible* $(l \to r)$*-substitution, and* q *a state of* $\mathcal{A}$. *We denote by* $\text{Norm}_\gamma(l \to r, \sigma, q)$ *the following set of transitions, called* normalization *of* $(l \to r, \sigma, q)$*:*

$$\begin{aligned}\{f(q_1, \ldots, q_k) \to q' \mid & p \in \mathcal{P}\text{os}_{\mathcal{F}}(r),\ t(p) = f,\\ & q' = q \text{ if } p = \varepsilon \text{ otherwise } q' = \gamma(l \to r, \sigma, q)(p)\\ & q_i = \gamma(l \to r, \sigma, q)(p.i) \text{ if } p.i \notin \mathcal{P}\text{os}_{\mathcal{X}}(r),\\ & \quad q_i = \sigma(\min\{p' \in \mathcal{P}\text{os}_{\mathcal{X}}(l) \mid l(p') = r(p.i)\}) \text{otherwise}\}\end{aligned}$$

The min *is computed for the lexical order.*

Notice that the set $\{p' \in \mathcal{P}\mathrm{os}_{\mathcal{X}}(l) \mid l(p') = r(p.i)\}$ used in the above definition is not empty. Indeed, in a term rewriting system variables occurring in the right-hand side must, by definition, occur in the left-hand side too.

Example 4 *Following Example 3, ε is the unique functional position of $r = h(x, y)$. Consequently, we set q' of the definition to be equal to q_f. Thus* $\mathrm{Norm}_{\gamma_{\mathrm{exe}}}(l \to r, \sigma_{\mathrm{exe}}, q_f)$ *is of the form $\{A \to q?, h(q?, q??) \to q_f\}$. Since for r, the position 1 is a functional position and 2 is in $\mathcal{P}\mathrm{os}_{\mathcal{X}}(r)$, we use the last line of the definition to compute $q??$ and $q?$ is defined by the approximation function γ_{exe}. Finally we obtain:*

$$\begin{aligned}\mathrm{Norm}_{\gamma_{\mathrm{exe}}}(l \to r, \sigma_{\mathrm{exe}}, q_f) &= \{r(1) \to \gamma_{\mathrm{exe}}(1), r(\varepsilon)(\gamma_{\mathrm{exe}}(1), \sigma_{\mathrm{exe}}(1)) \to q_0\} \\ &= \{A \to q_0, h(q_0, q_f) \to q_f\}.\end{aligned}$$

Lemma 1. *Let $\mathcal{A} = (\mathcal{Q}_0, \Delta, F_0)$ be a finite tree automaton, γ an approximation function, $l \to r \in \mathcal{R}$, σ an $\mathcal{A}$-compatible $(l \to r)$-substitution, and q a state of $\mathcal{A}$. If $l\sigma \to^*_{\mathcal{A}_0} q$ then*

$$r\sigma \to^*_{\mathrm{Norm}_\gamma(l \to r, \sigma, q)} q.$$

Proof is obvious. The transitions in Norm_γ are precisely added to reduce $r\sigma$ to q.

3.2 Completions

This section is dedicated to the proof of the main result: how to build a regular over-approximation of $\mathcal{R}^*(\mathcal{A})$? The above lemma shows how to over-approximate one rewriting step.

Lemma 2. *Let $\mathcal{A}_0 = (\mathcal{Q}_0, \Delta_0, F_0)$ be a finite tree automaton and γ an approximation function for $\mathcal{A}_0$. The automaton $\mathcal{C}_\gamma(\mathcal{A}_0) = (\mathcal{Q}_1, \Delta_1, F_1)$ is defined by:*

$$\Delta_1 = \bigcup \mathrm{Norm}_\gamma(l \to r, \sigma, q)$$

where the union involves all rules $l \to r \in \mathcal{R}$, all states $q \in \mathcal{Q}_0$, all $\mathcal{A}_0$-compatible $(l \to r)$-substitutions σ such that $l\sigma \to^*_{\mathcal{A}_0} q$ and $r\sigma \not\to^*_{\mathcal{A}_0} q$,

$$F_1 = F_0 \quad \textit{and} \quad \mathcal{Q}_1 = \mathcal{Q}_0 \cup \mathcal{Q}_1,$$

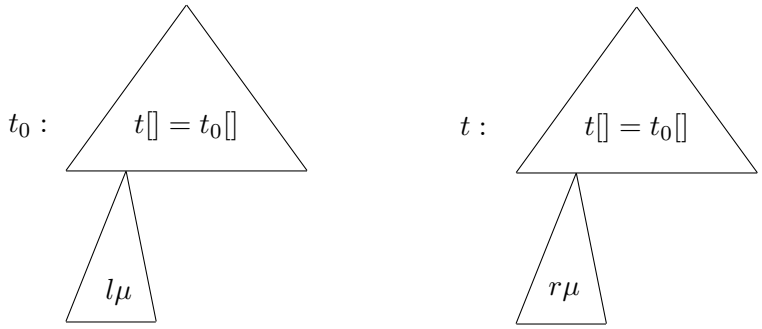
where $\mathcal{Q}_1$ denotes the set of states occurring in left or right-hand sides of transitions of Δ_1. One has

$$\mathcal{L}(\mathcal{A}_0) \cup \mathcal{R}(\mathcal{L}(\mathcal{A}_0)) \subseteq \mathcal{L}(\mathcal{C}_\gamma(\mathcal{A}_0)).$$

Proof. Let $t \in \mathcal{L}(\mathcal{A}_0) \cup \mathcal{R}(\mathcal{L}(\mathcal{A}_0))$. By definition of $\mathcal{C}_\gamma(\mathcal{A}_0)$ one has $\mathcal{L}(\mathcal{A}_0) \subseteq \mathcal{L}(\mathcal{C}_\gamma(\mathcal{A}_0))$. Consequently, if $t \in \mathcal{L}(\mathcal{A}_0)$, one has $t \in \mathcal{L}(\mathcal{C}_\gamma(\mathcal{A}_0))$. Thus we may now assume that

$t \in \mathcal{R}(\mathcal{L}(\mathcal{A}_0))$. Thus there exists a rule $l \to r \in \mathcal{R}$ a term t_0 in $\mathcal{L}(\mathcal{A}_0)$, a position p of t_0 and a substitution μ in $\Sigma(\mathcal{X}, \mathcal{T}(\mathcal{F}))$ such that

$$t_{0|p} = l\mu \quad \text{and} \quad t = t_0[r\mu]_p. \tag{1}$$



Since $t_0 \in \mathcal{L}(\mathcal{A}_0)$, there exists a state $q \in \mathcal{Q}_0$ and a state $q_f \in F_0$ such that

$$l\mu \to^*_{\mathcal{A}_0} q \quad \text{and} \quad t_0[q]_p \to^*_{\mathcal{A}_0} q_f. \tag{2}$$

Since $l\mu \to^*_{\mathcal{A}_0} q$ there exists an $(l \to r)$-substitution σ such that $l\mu \to_{\mathcal{A}_0} l\sigma$. Furthermore, for each $x \in \mathcal{V}\mathrm{ar}(l)$,

$$\mu(x) \in \bigcap_{p \in \mathcal{P}\mathrm{os}_{\{x\}}(l)} \mathcal{L}(\mathcal{A}, \sigma(p)),$$

thus the $(l \to r)$-substitution σ is $\mathcal{A}_0$ compatible. Therefore, using Lemma 1, one has

$$r\sigma \to^*_{\mathcal{C}_\gamma(\mathcal{A}_0)} q. \tag{3}$$

For each variable x occurring in l and all positions p of x in l one has $\mu(x) \to^*_{\mathcal{A}_0} \sigma(p)$. In particular, for each variable x occurring in l, $\mu(x) \to^*_{\mathcal{A}_0} \sigma(p')$, where p' is the minimal position where x occurs in l. Consequently and by definition of $r\sigma$, one has

$$r\mu \to^*_{\mathcal{A}_0} r\sigma. \tag{4}$$

We are now able to conclude.

$$\begin{array}{lll} t = & t_0[r\mu] & \text{using (1)} \\ \to^*_{\mathcal{A}_0} & t_0[r\sigma] & \text{using (4)} \\ \to^*_{\mathcal{C}_\gamma(\mathcal{A}_0)} & t_0[q] & \text{using (3)} \\ \to^*_{\mathcal{A}_0} & q_f & \text{using (2)} \end{array}$$

Thus $t \in \mathcal{L}(\mathcal{C}_\gamma(\mathcal{A}_0))$, proving the theorem.

Let us remark that using well chosen approximation functions may iteratively lead to a fixpoint automaton which recognizes an over-approximation of $\mathcal{R}^*(\mathcal{A}_0)$. One can formally express this by the following (soundness) main theorem.

Theorem 5 *Let $(\mathcal{A}_n)$ and (γ_n) be respectively a sequence a finite tree automata and a sequence of approximation functions defined by: for each integer n, γ_n is an approximation function for $\mathcal{A}_n$ and*

$$\mathcal{A}_{n+1} = \mathcal{C}_{\gamma_n}(\mathcal{A}_n).$$

If the sequence $(\mathcal{A}_n)$ is ultimately constant equal to $\mathcal{B}$, then

$$\mathcal{R}^*(\mathcal{L}(\mathcal{A}_0)) \subseteq \mathcal{L}(\mathcal{B}).$$

The proof is immediate by a simple induction using Lemma 2. Notice that in [6], we have defined a family of approximation functions which can be automatically generated. Another advantage is that they can be easily adapted to the present construction as explained in the next section. Furthermore using appropriate approximation functions may lead to a proof of the non-reachability of a term. However, these methods don't provide a way to prove that a particular term is reachable, since we compute an over-approximation of reachable terms (the method is not complete).

An example is given in Appendix A.

4 Application to the *View-Only* Protocol

In this section we illustrate the main concepts on the example of the *view-only* protocol, and we explain how to manipulate the tool `TA4SP` supporting an automated verification.

4.1 The *View-Only* Protocol

Before describing the protocol, notice that encoding security protocols and secrecy properties with tree automata and term rewriting systems is classical [].

The *View-Only* protocol (Fig. 2) is a component of the *Smartright* system [1]. In the context of home digital network, this system prevents users from unlawfully copying movies broadcast on the network. The *view-only* participants are a decoder (`DC`) and a digital TV set (`TVS`). They share a secret key `Kab` securely sealed in both of them. The goal of this protocol is to periodically change a secret control word (`CW`) enabling to decode the current broadcast program. As seen in Fig. 2, the properties of the `XOR` operator allow to establish the sharing of `CW` between the participants.

The data `VoKey`, `VoR` and `VoRi` are randomly generated numbers. The functional symbol `h` represents a one-way function, meaning that no-one can guess `x` from `h(x)`, unless he already knows `x`.

Let us explain how this protocol works.

- **Step 1:** `DC` sends a message containing `xor(CW,VoR)` and `VoKey` to `TVS`. This message is encoded by the private shared key `Kab`. The data `VoKey` is a fresh symmetric key used along the session. At this stage, `TVS` can extract neither `CW` nor `VoR` from `xor(CW,VoR)` since `TVS` knows neither `CW` nor `VoR`.

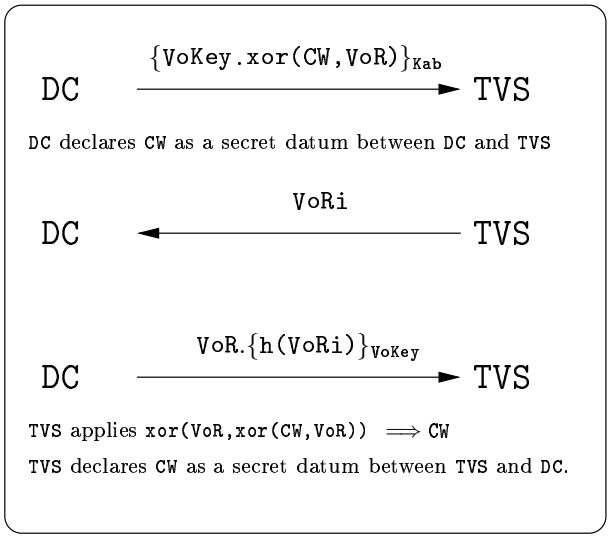


Fig. 2. The *"view-only"* Protocol

- **Step 2:** TVS sends in return a random challenge VoRi whose goal is to identify DC.
- **Step 3:** TVS replies by sending $\texttt{VoR.\{h(VoRi)\}}_{\texttt{VoKey}}$. Receiving this message, TVS both checks whether the challenge's answer is correct (by comparing the hashed value h(VoRi) with its own value), and extracts CW from the xored datum xor(CW,VoR) received at step 1 and using VoR. This is done by computing xor(xor(CW,VoR),VoR), and by applying sequentially rules II., IV. and III. of Fig. 1 to it, TVS obtains: xor(xor(CW,VoR), VoR) $\stackrel{\texttt{II.}}{\rightarrow}$ xor(CW,xor(VoR,VoR)) $\stackrel{\texttt{IV.}}{\rightarrow}$ xor(CW,0) $\stackrel{\texttt{III.}}{\rightarrow}$ CW.

  Notice that Rule IV. of Fig. 1 is crucial at Step 3 of this protocol.

We have implemented our approach within the TA4SP tool presented in the following subsection.

4.2 Using TA4SP

This tool, whose method is detailed in [6], is one of the four official tools of the AVISPA tool-set [2]. The particularity of this tool is verifying of secrecy properties for an unbounded number of sessions.

The structure of the TA4SP tool is detailed in Fig. 3.

The language IF is a low level specification language automatically generated from HLPSL (High Level Protocol Specification Language) [9] in the AVISPA toolset.

The TA4SP tool is made up of:

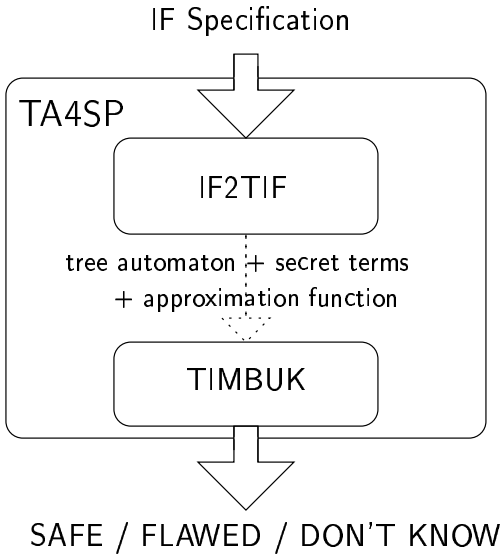


Fig. 3. `TA4SP` tool

- `IF2TIF`, a translator from `IF` to a specification well-adapted to `TIMBUK+`, and
- `TIMBUK+`,[2] a collection of tools for achieving proofs of reachability over term rewriting systems and for manipulating tree automata. This tool has been initially developed by Th. Genet (IRISA/ INRIA-Rennes, FRANCE) and improved to handle our approximation functions.

Let us remark that the available version of `TA4SP` at http://www.avispa-project.org used in the framework of the AVISPA project is not yet updated with the `XOR` features. It is intended that this be updated in the near future.

4.3 Verifying the *View-Only* Protocol

In [18], the authors verified that no old value of `CW` can be reused. Indeed, if the freshness of `CW` was not satisfied then we can imagine that the copy-protection would become obsolete. By storing all control words and by reusing them, an unethical individual could for instance decrypt the broadcasted program without paying the amount. However, the model considered is strongly typed in the sense that the authors handle the `XOR` operator only for terms satisfying a particular given form.

In order to consider type confusing attacks, we have succeeded in verifying the secrecy of `CW` on an untyped model for the `XOR` algebraic properties, using the method developed in this paper. Furthermore, using the family of approximation functions defined in [6] we have succeeded in verifying it automatically.

[2] Timbuk is available at http://www.irisa.fr/lande/genet/timbuk/.

Time computation of the verification is about one hundred minutes on a classical laptop, but we hope to manage to have a faster computation by removing some redundant calculus. The computed fixed point automaton (encoding an over-approximation of intruder's knowledge) has 203 states and 583 transitions. Some technical details (HLPSL specification, input/output automata, etc), on this verification are given in Appendix B.

5 Conclusion

This paper shows that the symbolic approximation-based approach we have been developing is well-adapted for analyzing security protocols using algebraic properties while considering an unbounded number of sessions. Indeed, the automatically generated symbolic approximation function enables us 1) an automated normalization of transitions, and 2) an automated completion procedure. Notice that the variables used to manipulate algebraic properties are not typed, like in [23]. Our symbolic approximation-based framework allowing us to handle algebraic properties does not deal with timestamps.

The tool `TA4SP` has been updated to take the exclusive or algebraic property of cryptographic primitives into account. This way the feasibility of the analysis has been confirmed by the experimentation on the *view-only* protocol. Future development concerns the implementation optimization.

We intend to investigate further algebraic properties that can be handled in practice. We anticipate that it could be carried out for algebraic properties expressed by quadratic rules. At this stage, experiments should be performed again.

To the best of our knowledge, this is the first attempt to automatically handle a large class of algebraic properties used in cryptographic protocols. Indeed, we wish to emphasize the fact that our theoretical framework is supported by a *push-button* tool `TA4SP` [3,5]. Moreover, `TA4SP` is used for protocols specified in the standard High Level Protocol Specification Language (HLPSL) [9,4]. This language is known to be suitable for industrial users.

These two significant advantages make it possible to use our framework and the fully automatic tool in the industrial context.

References

1. Smartright technical white paper v1.0. Technical report, Thomson, http://www.smartright.org, October 2001.
2. A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. Hankes Drielsma, P.-C. Héam, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santos Santiago, M. Turuani, L. Viganò, and L. Vigneron. The AVISPA Tool for the automated validation of internet security protocols and applications. In K. Etessami and S. Rajamani, editors, *17th International Conference on Computer Aided Verification, CAV'2005*, volume 3576 of *Lecture Notes in Computer Science*, pages 281–285, Edinburgh, Scotland, 2005. Springer.

3. A. Armando, D. Basin, M. Bouallagui, Y. Chevalier, L. Compagna, S. Mödersheim, M. Rusinowitch, M. Turuani, L. Viganò, and L. Vigneron. The AVISS Security Protocol Analysis Tool. In *Proceedings of CAV'02*, LNCS 2404, pages 349–354. Springer, 2002.
4. AVISPA. Deliverable 2.1: The High-Level Protocol Specification Language. Available at *http://www.avispa-project.org, 2003.*
5. *AVISPA. Deliverable 7.2: Assessment of the AVISPA tool v.1. Available at http://www.avispa-project.org, 2003.*
6. *Y. Boichut, P.-C. Héam, and O. Kouchnarenko. Automatic Verification of Security Protocols Using Approximations. Research Report RR-5727, INRIA-Lorraine - CASSIS Project, October 2005.*
7. *Y. Boichut, P.-C. Héam, O. Kouchnarenko, and F. Oehl. Improvements on the Genet and Klay technique to automatically verify security protocols. In* Proc. Int. Ws. on Automated Verification of Infinite-State Systems (AVIS'2004), joint to ETAPS'04, *pages 1–11, Barcelona, Spain, April 2004. The final version will be published in EN in Theoretical Computer Science, Elsevier.*
8. *L. Bozga, Y. Lakhnech, and M. Perin. HERMES: An automatic tool for verification of secrecy in security protocols. In* CAV: International Conference on Computer Aided Verification, *2003.*
9. *Y. Chevalier, L. Compagna, J. Cuellar, P. Hankes Drielsma, J. Mantovani, S. Mödersheim, and L. Vigneron. A high level protocol specification language for industrial security-sensitive protocols. In* Proceedings of Workshop on Specification and Automated Processing of Security Requirements (SAPS), *volume 180, Linz, Austria, September 2004. Oesterreichische Computer Gesellschaft (Austrian Computer Society).*
10. *Y. Chevalier, R. Kusters, M. Rusinowitch, and M. Turuani. An NP decision procedure for protocol insecurity with XOR.* TCS: Theoretical Computer Science, *338, 2005.*
11. *I. Cibrario, L. Durante, R. Sisto, and A. Valenzano. Automatic detection of attacks on cryptographic protocols: A case study. In Christopher Kruegel Klaus Julisch, editor,* Intrusion and Malware Detection and Vulnerability Assessment: Second International Conference, *volume 3548 of* Lecture Notes in Computer Science, *Vienna, 2005.*
12. *H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications, 2002. http://www.grappa.univ-lille3.fr/tata/.*
13. *H. Comon-Lundh and V. Cortier. Tree automata with one memory, set constraints and cryptographic protocols.* Theoretical Computer Science, *331(1):143–214, February 2005.*
14. *V. Cortier, S. Delaune, and P. Lafourcade. A survey of algebraic properties used in cryptographic protocols.* Journal of Computer Security, *2005.*
15. *N. A. Durgin, P. D. Lincoln, J. C. Mitchell, and A. Scedrov. Undecidability of bounded security protocols. In* Proc. Int. Ws. on Formal Methods and Security Protocols, *Italy, Trento, August 1999.*
16. *G. Feuillade, Th Genet, and V. Viet Triem Tong. Reachability analysis of term rewriting systems.* Journal of Automated Reasonning, *33, 2004.*
17. *Th. Genet and F. Klay. Rewriting for cryptographic protocol verification. In* Proc. Int. Conf. CADE'00, *LNCS 1831, pages 271–290. Springer-Verlag, 2000.*
18. *Th. Genet, Y.-M. Tang-Talpin, and V. Viet Triem Tong. Verification of copy-protection cryptographic protocol using approximations of term rewriting systems. In* Proc. of WITS'03, Workshop on Issues in the Theory of Security, *2003.*
19. *D. Monniaux. Abstracting cryptographic protocols with tree automata. In* Sixth International Static Analysis Symposium (SAS'99), *number 1694 in Lecture Notes in Computer Science. Springer-Verlag, 1999.*
20. *H. Ohsaki and T. Takai. Actas: A system design for associative and commutative tree automata theory. In* Proc. of the 5th Int. Ws. on Rule-Based Programming: RULE'2004, *Aachen, Germany, June 2004. To appear in ENTCS.*
21. *M. Rusinowitch and M. Turuani. Protocol insecurity with finite number of sessions is NP-complete. In* 14th IEEE Computer Security Foundations Workshop (CSFW '01), *pages 174–190, Washington - Brussels - Tokyo, June 2001. IEEE.*

22. *T. Truderung. Regular protocols and attacks with regular knowledge. In* Proc. of 20th Int. Conf. on Automated Deduction (CADE'05), *volume 3632 of* LNCS, *pages 377–391. Springer, 2005.*
23. *R. Zunino and P. Degano. Handling exp, x (and timestamps) in protocol analysis. To appear in Proc. of Int. Conf. FOSSACS'06, 2006.*

Appendix A: Completion Example

In this section we explain how our approach works on a toy example . We do not give the details of a protocol study since involving term rewriting systems are too huge to be readable.

We consider terms defined by

- $\mathcal{F}_0 = \{0\}$,
- $\mathcal{F}_1 = \{\text{Inv}, s\}$,
- $\mathcal{F}_2 = \{+\}$ and
- $\mathcal{F}_{k\geq 3} = \emptyset$.

In this formalism, the symbol s denotes the successor function. For instance, $s(s(s(0)))$ is the successor of the successor of the successor of 0 and denotes the integer 3. The operator Inv denotes the inverse (for the addition). For example, $\text{Inv}(s(0))$ is the inverse of the successor of 0 and denotes the integer -1.

We use the following term rewriting system to encode addition and subtraction over $\mathbb{Z}$. To simplify notations, we write $(x + y)$ or $x + y$ for $+(x, y)$.

$$\begin{aligned}
\mathcal{R} = \{&\text{Inv}(\text{Inv}(x)) \to x && (5)\\
&x \to \text{Inv}(\text{Inv}(x)) && (6)\\
&x + \text{Inv}(x) \to 0 && (7)\\
&x + y \to y + x && (8)\\
&x + (y + z) \to (x + y) + z && (9)\\
&x + 0 \to x && (10)\\
&x + s(0) \to s(x) && (11)\\
&s(x) \to x + s(0) && (12)\\
&\text{Inv}(s(x)) \to \text{Inv}(s(s(x))) + s(0)\} && (13)
\end{aligned}$$

Notice that this term rewriting system is not left-linear (Rule (7)).

We are interested in the following problem: given three integers a, b and c, are there integers λ and μ such that

$$\lambda a + \mu b = c?$$

A basic number theory result states that the answer the previous question is yes if and only if c is a multiple of the greatest common divisor of a and b.

For instance, it is possible for $a = 7$, $b = 3$ and $c = 15$ (since $\gcd(a, b) = 1$). We may prove it using the above term rewriting system. Indeed, from $s^7(0)$ and $s^3(0)$ one can reach $s^{15}(0)$ using $+$, Inv and rewriting rules. For example:

$$s^3(0) \to_{12} s^2(0) + s(0) \to_{12} (s(0) + s(0)) + s(0)$$

Consequently

$$s^3(0) + s^3(0) \to_{12}^* ((s(0) + s(0)) + s(0)) + s^3(0) \to_{12,9}^* s^6(0) \tag{14}$$

Similarly one has

$$(((s^7(0) + s^7(0)) + s^7(0)) \to_{12,9} s^2 1(0) \tag{15}$$

Moreover, from (14) one has

$$\mathrm{Inv}(s^3(0) + s^3(0)) \to_{12,9}^* \mathrm{Inv}(s^6(0)) \to_{9,8}^* \mathrm{Inv}(s^{21}(0)) + s^{15}(0)$$

Therefore, by (15), one has

$$(((s^7(0)+s^7(0))+s^7(0))+\mathrm{Inv}(s^3(0)+s^3(0)) \to_{8,12,9}^* (s^2 1(0)+\mathrm{Inv}(s^{21}(0)))+s^{15}(0) \to_{7,10} s^{15}(0).$$

Now we prove that the problem has no solution for $a = 2$, $b = 4$ and $c = 5$ (this is mathematically trivial, the goal is just to illustrate that it can be proved automatically by our over-approximation approach).

We consider for initial terms the language accepted by the following tree automaton $\mathcal{A}$:

- States are q_0, q_1, q_2, q_3, q_4, q_{-2} q_{-4} and q_f,
- Final states are q_2,q_{-2}, q_{-4}, q_4, and q_f,
- Transitions are
  - $0 \to q_0$, $s(q_0) \to q_1$, $s(q_1) \to q_2$, $s(q_2) \to q_3$, $s(q_3) \to q_4$ (encodes that $s^2(0)$ and $s^4(0)$ are initially known),
  - $\mathrm{Inv}(q_4) \to q_{-4}$ (encodes that one can compute the inverse of 4),
  - $\mathrm{Inv}(q_2) \to q_{-2}$ (encodes that one can compute the inverse of 2),
  - $q_{f_1}+q_{f_2} \to q_f$ for all final states q_{f_1}, q_{f_2}, (encodes that one can do the addition of two computed integers terms).

We want to prove that $s^5(0) \notin \mathcal{R}^*(\mathcal{L}(\mathcal{A}))$.
We give some details on the first completion step.

Rule (5) This rule doesn't provide new transition. Indeed, there is no state q in $\mathcal{A}$ such that $\mathrm{Inv}(\mathrm{Inv}(q))$ can be derivated in $\mathcal{A}$ to a state.

Rule (6) For each state q one has to add the normalisation of the transition $\mathrm{Inv}(\mathrm{Inv}(q)) \to q$. Assume that

$$\gamma(Rule(6), \{\varepsilon \mapsto q_1\}, q_1)(1) = q_3$$

. Then during the completion step, the normalisation of $\mathrm{Inv}(\mathrm{Inv}(q_1)) \to q_1$ ensures that we add the transitions $\mathrm{Inv}(q_1) \to q_3$ and $\mathrm{Inv}(q_3) \to q_1$. With similar assumptions on γ one adds during the first completion step $\mathrm{Inv}(q_0) \to q_0$, $\mathrm{Inv}(q_{-4} \to q_4)$ and $\mathrm{Inv}(q_{-2}) \to q_2$.

Rule (7) Since $q_4 + \mathrm{Inv}(q_4) \to^*_{\mathcal{A}} q_f$, one has to add the transition $0 \to q_f$.
Rule (8) This rule doesn't provide new transition.
Rule (9) This rule doesn't provide new transition.
Rule (10) This rule doesn't provide new transition.
Rule (11) This rule doesn't provide new transition.
Rule (12) Since $s(q_0) \to_{\mathcal{A}} q_1$ and $q_0 + s(0) \not\to^*_{\mathcal{A}} q_1$, one has to add the following transitions (with correct assumptions on γ) $0 \to q_0$, $s(q_0) \to q_1$ (these two transitions are already in $\mathcal{A}$) and $q_0 + q_1 \to q_1$. Similarly, one has to add transitions $q_0 + q_2 \to q_2$, $q_0 + q_3 \to q_3$, $q_0 + q_4 \to q_4$.
Rule (12) Since $\mathrm{Inv}(s(q_1)) \to^*_{\mathcal{A}} q_{-2}$ and $\mathrm{Inv}(s(s(q_1)) + s(0)) \not\to^*_{\mathcal{A}} q_{-2}$, one has to add the transitions (with correct assumption on γ), $s(0) \to q_1$, $s(q_1) \to q_2$, $s(q_2) \to q_3$, $\mathrm{Inv}(q_3) \to q_1$, $q_1 + q_1 \to q_2$ and $\mathrm{Inv}(q_2) \to q_{-2}$.

Similar completion steps lead to the following tree automaton $\mathcal{B}$:

- States of $\mathcal{B}$ are $q_{-4}, q_{-2}, q_1, q_2, q_3, q_4$ and q_f.
- Final states are q_2, q_4, q_{-2}, q_{-4} and q_f.
- Transitions on constants are $0 \to q_0$ and $0 \to q_f$.
- Transitions with symbol s are given by the following table:

	q_0	q_1	q_2	q_3	q_4
s	q_1	q_2	q_3	q_4	q_1

  For instance, $s(q_{-2}) \to q_3$ is a transition.
- Transitions with symbol Inv are given by the following table:

	q_{-4}	q_{-2}	q_0	q_1	q_2	q_3	q_4	q_f
Inv	q_4	q_2	q_0	q_3	q_{-2}	q_1	q_{-4}	q_f

- Transitions with symbol $+$ are given by the following table:

$+$	q_{-4}	q_{-2}	q_0	q_1	q_2	q_3	q_4	q_f
q_{-4}	q_{-4}, q_f	q_{-2}, q_f	q_{-4}	q_1	q_2, q_f	q_3	q_0, q_4, q_f	q_f
q_{-2}	q_{-2}, q_f	q_0, q_f	q_{-2}, q_f	q_3	q_0, q_4, q_f	q_1	q_2, q_f	q_f
q_0	q_{-4}, q_f	q_{-2}, q_f	q_0	q_1	q_2, q_f	q_3	q_4, q_f	$\emptyset$
q_1	q_1	q_3	q_1	q_2, q_f	q_3	q_4, q_0, q_f	q_f	$\emptyset$
q_2	q_2, q_f	q_0, q_4, q_f	q_2, q_f	q_3	q_4, q_f, q_0	q_1	q_2, q_f	q_f
q_3	q_3	q_1	q_3	q_4, q_0, q_f	q_1	q_2, q_f	q_3	$\emptyset$
q_4	q_0, q_4, q_f	q_2, q_f	q_4, q_0, q_f	q_1	q_2, q_f	q_3	q_4, q_f, q_0	q_f
q_f	q_f	q_f	$\emptyset$	$\emptyset$	q_f	$\emptyset$	q_f	q_f

The automaton $\mathcal{B}$ is stable by the $\mathcal{C}_\gamma$ completion. Consequently, it accepts an over-approximation of reachable terms of $\mathcal{A}$ by $\mathcal{R}$. Since $s^5(0) \notin \mathcal{L}(\mathcal{B})$, its proved that we may not have $\lambda.2 + \mu.4 = 5$ with $\lambda, \mu \in \mathbb{Z}$.

Appendix B: Technical Details on the Protocol Verification

In this section we give some technical issues on the verification of the *View only protocol*.

The HLPSL Specification of the Protocol

This subsection is dedicated to the formal specification of the *View only protocol* in HLPSL (High Level Specification Language). Since HLPSL is not the topic of this paper, the interested reader may refer [4] for more detail.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% HLPSL:
role alice (A,B     : agent,
            Ks    : symmetric_key,
            H: hash_func,
            Snd,Rcv : channel (dy)) played_by A def=

  local
     State  : nat,
     VoKey  : text,
     VoR    : text,
     CW     : symmetric_key,
     VoRi   : text,
     SSet   : agent set

  init State := 0 /\ SSet := {A,B}

  transition
    1. State=0 /\ Rcv(start) =|>
          VoKey' := new()
       /\ VoR':=new()
       /\ CW':=new()
       /\ Snd({VoKey'.xor(CW',VoR')}_Ks)
       /\ State' := 1
       /\ secret(CW',sna,SSet)


    2. State=1 /\ Rcv(VoRi') =|>
          Snd(VoR.{H(VoRi')}_VoKey)
```

```
       /\ State' := 2


end role

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

role bob (B,A     : agent,
          Ks    : symmetric_key,
  H: hash_func,
          Snd,Rcv : channel (dy)) played_by B def=

  local
     State : nat,
     VoKey,VoR    : text,
     CW : symmetric_key,
     X,Y:       message,
     VoRi    : text ,
     SSet  : agent set

  init State := 0 /\ SSet := {A,B}

  transition
    1. State=0 /\ Rcv({VoKey'.X'}_Ks) =|>
          State' := 1
       /\ VoRi' := new()
       /\ Snd(VoRi')


    2. State=1 /\ Rcv(VoR'.{H(VoRi)}_VoKey)
        =|>  Y'=xor(VoR',X)
          State' := 2
/\ secret(Y',snb,SSet)


end role

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

role session (A,B: agent,
              Ks : symmetric_key,
```

```
              H:    hash_func) def=

  local SA, RA, SB, RB: channel (dy)

  composition
    alice (A,B,Ks,H,SA,RA) /\
    bob   (B,A,Ks,H,SB,RB)

end role

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

role environment() def=

  const
     a, b, i    : agent,
     kab, kbi, kai : symmetric_key,
     na, nb,
     sna, snb   : protocol_id,
     h          : hash_func

  intruder_knowledge = {a,b,i,kai,kbi,h}

  composition
    session(a,b,kab,h) /\
    session(a,i,kai,h) /\
    session(b,i,kbi,h) /\
    session(i,a,kai,h) /\
    session(i,b,kbi,h)



end role

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

goal
     secrecy_of sna, snb
end goal
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
environment()
```

Timbuk Input Format of the Protocol

We provide now the Term Rewriting System deduced from the above specification in the Timbuk Input Format. For more information on this format, the reader is refereed to [16].

```
Ops secret0sna:5 secret3snb:5 func:1 sk:1 set_84:0 set_81:0 set_77:0 kbi:0
set_74:0 kai:0 i:0 set_69:0 dummy_msg:0 set_60:0 dummy_nonce:0 h:0 kab:0 b:0
a:0 text:1 agt:1 msg:1 nat:1 nil:0 snb:0 state_bob:12 apply:2 deux:0
state_alice:11 iknows:1 xor:2 pair:2 scrypt:2 start:0 sna:0 un:0 default34:3
default33:3 default32:3 default31:3 zeros:0 default30:3 nonce:1 inv:1 e:0



Vars Xdefault34 Xdefault30 Xdefault31 Xdefault32 Dummy_CWstate_bob
Dummy_VoRstate_bob Xstate_bob VoKeystate_bob Dummy_VoRistate_alice
SSetstate_alice VoRistate_alice Dummy_CWstate_alice Dummy_VoRstate_alice
Dummy_VoKeystate_alice Hstate_alice Ksstate_alice SSetstate_bob
Dummy_VoRistate_bob Dummy_Xstate_bob CWstate_bob VoRstate_bob
Dummy_VoKeystate_bob Hstate_bob Ksstate_bob Bstate_bob Astate_bob
Astate_alice Bstate_alice x65 x64 x63 x61 x60 x59 x58 x56 x55 delphine x53
x52 x51 x50 x49 x48 x47 x46 x45 x44 x43 x42 x41 x40 x39 x38 x37 x36 x35 x34
x33 x32 x31 x30 x29 x28 x27 x26 x25 x24 x23 x22 x21 x20 x19 x18 x17 x16 x15
x14 x13 x12 x11 x10 x9 x8 x7 x6 x5 x4 x3 x2 x1 x0 alpha x y z

TRS TermrewritingSystem

state_alice(agt(Astate_alice), agt(Bstate_alice), sk(Ksstate_alice),
func(Hstate_alice), nat(zeros), text(Dummy_VoKeystate_alice),
text(Dummy_VoRstate_alice), text(Dummy_CWstate_alice),
text(VoRistate_alice), SSetstate_alice, iknows(msg(start))) ->
state_alice(agt(Astate_alice), agt(Bstate_alice), sk(Ksstate_alice),
func(Hstate_alice), nat(un), nonce(default30(Astate_alice, Bstate_alice,
zeros)), nonce(default32(Astate_alice, Bstate_alice, zeros)),
nonce(default31(Astate_alice, Bstate_alice, zeros)), text(VoRistate_alice),
SSetstate_alice,
iknows(scrypt(sk(Ksstate_alice),pair(nonce(default30(Astate_alice,
Bstate_alice, zeros)),xor(nonce(default31(Astate_alice, Bstate_alice,
zeros)),nonce(default32(Astate_alice, Bstate_alice, zeros)))))))

state_bob(agt(Bstate_bob), agt(Astate_bob), sk(Ksstate_bob),
func(Hstate_bob), nat(zeros), text(Dummy_VoKeystate_bob),
text(VoRstate_bob), text(CWstate_bob), Dummy_Xstate_bob,
text(Dummy_VoRistate_bob), SSetstate_bob,
iknows(scrypt(sk(Ksstate_bob),pair(nonce(Xdefault30),xor(nonce(Xdefault31),nonce(Xdefault32))))))
-> state_bob(agt(Bstate_bob), agt(Astate_bob), sk(Ksstate_bob),
```

```
func(Hstate_bob), nat(un), nonce(Xdefault30), text(VoRstate_bob),
text(CWstate_bob), xor(nonce(Xdefault31),nonce(Xdefault32)),
nonce(default34(Bstate_bob, Astate_bob, zeros)), SSetstate_bob,
iknows(nonce(default34(Bstate_bob, Astate_bob, zeros))))

state_alice(agt(Astate_alice), agt(Bstate_alice), sk(Ksstate_alice),
func(Hstate_alice), nat(un), nonce(default30(Astate_alice, Bstate_alice,
zeros)), nonce(default32(Astate_alice, Bstate_alice, zeros)),
nonce(default31(Astate_alice, Bstate_alice, zeros)), text(VoRistate_alice),
SSetstate_alice, iknows(nonce(Xdefault34))) ->
state_alice(agt(Astate_alice), agt(Bstate_alice), sk(Ksstate_alice),
func(Hstate_alice), nat(deux), nonce(default30(Astate_alice, Bstate_alice,
zeros)), nonce(default32(Astate_alice, Bstate_alice, zeros)),
nonce(default31(Astate_alice, Bstate_alice, zeros)), nonce(Xdefault34),
SSetstate_alice, iknows(pair(nonce(default32(Astate_alice, Bstate_alice,
zeros)),scrypt(nonce(default30(Astate_alice, Bstate_alice,
zeros)),apply(func(Hstate_alice),nonce(Xdefault34)))))

state_bob(agt(Bstate_bob), agt(Astate_bob), sk(Ksstate_bob),
func(Hstate_bob), nat(un), nonce(Xdefault30), text(VoRstate_bob),
text(CWstate_bob), xor(nonce(Xdefault31),nonce(Xdefault32)),
nonce(default34(Bstate_bob, Astate_bob, zeros)), SSetstate_bob,
iknows(pair(nonce(Xdefault32),scrypt(nonce(Xdefault30),apply(func(Hstate_bob),
nonce(default34(Bstate_bob,Astate_bob, zeros)))))) ->
state_bob(agt(Bstate_bob),
agt(Astate_bob),
sk(Ksstate_bob), func(Hstate_bob), nat(deux), nonce(Xdefault30),
nonce(Xdefault32), nonce(Xdefault31),
xor(nonce(Xdefault32),xor(nonce(Xdefault31),nonce(Xdefault32))),
nonce(default34(Bstate_bob, Astate_bob, zeros)), SSetstate_bob,
iknows(text(nil)))

pair(nonce(Xdefault30),scrypt(nonce(Xdefault30),z)) -> z

pair(sk(Ksstate_alice),scrypt(sk(Ksstate_alice),z)) -> z


pair(x,y) -> x

pair(x,y) -> y

iknows(x) -> x


xor(x,y) -> e

xor(x,y) -> xor(y,x)

xor(x,xor(y,z)) -> xor(xor(x,y),z)

xor(x,e) -> x
```

Automaton etatInitial

States qi qagti qb qagt3 qa qagt5 qset_84 q7 qset_81 q9 qset_77 q11 qkbi q13
qset_74 q15 qkai q17 qset_69 q19 qdummy_msg q21 qset_60 q23 qdummy_nonce q25
qh q27 qkab q29 qnil q31 qsnb q33 qdeux q35 qstart q37 qsna q39 qun q41
qzeros q43 q44 q45 q46 q47 q48 q49 q50 q51 q52 q53 q54 q55 q56 q57 q58 q59
q60 q61 q62 q63 q64 q65 q66 q67 q68 q69 q70 q71 q72 q73 q74 q75 q76 q77 q78
q79 q80 q81 q82 q83 q84 q85 q86 q87 q88 q89 q90 q91 q92 q93 q94 q95 q96 q97
q98 q99 q100 q101 q102 q103 q104 q105 q106 q107 q108 q109 q110 q111 q112
q113 q114 q115 q116 q117 q118 q119 q120 q121 q122 q123 q124 q125 q126 q127
q128 q129 q130 q131 q132 q133 q134 q135 q138 q140 qstate qnet

Final States qnet qstate

Prior

i->qi
agt(qi)->qagti
b->qb
agt(qb)->qagt3
a->qa
agt(qa)->qagt5
set_84->qset_84
msg(qset_84)->q7
set_81->qset_81
msg(qset_81)->q9
set_77->qset_77
msg(qset_77)->q11
kbi->qkbi
sk(qkbi)->q13
set_74->qset_74
msg(qset_74)->q15
kai->qkai
sk(qkai)->q17
set_69->qset_69
msg(qset_69)->q19
dummy_msg->qdummy_msg
msg(qdummy_msg)->q21
set_60->qset_60
msg(qset_60)->q23
dummy_nonce->qdummy_nonce
text(qdummy_nonce)->q25
h->qh
func(qh)->q27
kab->qkab
sk(qkab)->q29
nil->qnil
text(qnil)->q31
snb->qsnb
nat(qsnb)->q33
deux->qdeux

```
nat(qdeux)->q35
start->qstart
msg(qstart)->q37
sna->qsna
nat(qsna)->q39
un->qun
nat(qun)->q41
zeros->qzeros
nat(qzeros)->q43
default30(qa, qa, qzeros)->q44
nonce(q44)->q45
default30(qa, qb, qzeros)->q46
nonce(q46)->q47
default30(qa, qi, qzeros)->q48
nonce(q48)->q49
default30(qb, qa, qzeros)->q50
nonce(q50)->q51
default30(qb, qb, qzeros)->q52
nonce(q52)->q53
default30(qb, qi, qzeros)->q54
nonce(q54)->q55
default30(qi, qa, qzeros)->q56
nonce(q56)->q57
default30(qi, qb, qzeros)->q58
nonce(q58)->q59
default30(qi, qi, qzeros)->q60
nonce(q60)->q61
default31(qa, qa, qzeros)->q62
nonce(q62)->q63
default31(qa, qb, qzeros)->q64
nonce(q64)->q65
default31(qa, qi, qzeros)->q66
nonce(q66)->q67
default31(qb, qa, qzeros)->q68
nonce(q68)->q69
default31(qb, qb, qzeros)->q70
nonce(q70)->q71
default31(qb, qi, qzeros)->q72
nonce(q72)->q73
default31(qi, qa, qzeros)->q74
nonce(q74)->q75
default31(qi, qb, qzeros)->q76
nonce(q76)->q77
default31(qi, qi, qzeros)->q78
nonce(q78)->q79
default32(qa, qa, qzeros)->q80
nonce(q80)->q81
default32(qa, qb, qzeros)->q82
nonce(q82)->q83
default32(qa, qi, qzeros)->q84
nonce(q84)->q85
default32(qb, qa, qzeros)->q86
```

```
nonce(q86)->q87
default32(qb, qb, qzeros)->q88
nonce(q88)->q89
default32(qb, qi, qzeros)->q90
nonce(q90)->q91
default32(qi, qa, qzeros)->q92
nonce(q92)->q93
default32(qi, qb, qzeros)->q94
nonce(q94)->q95
default32(qi, qi, qzeros)->q96
nonce(q96)->q97
default33(qa, qa, qzeros)->q98
nonce(q98)->q99
default33(qa, qb, qzeros)->q100
nonce(q100)->q101
default33(qa, qi, qzeros)->q102
nonce(q102)->q103
default33(qb, qa, qzeros)->q104
nonce(q104)->q105
default33(qb, qb, qzeros)->q106
nonce(q106)->q107
default33(qb, qi, qzeros)->q108
nonce(q108)->q109
default33(qi, qa, qzeros)->q110
nonce(q110)->q111
default33(qi, qb, qzeros)->q112
nonce(q112)->q113
default33(qi, qi, qzeros)->q114
nonce(q114)->q115
default34(qa, qa, qzeros)->q116
nonce(q116)->q117
default34(qa, qb, qzeros)->q118
nonce(q118)->q119
default34(qa, qi, qzeros)->q120
nonce(q120)->q121
default34(qb, qa, qzeros)->q122
nonce(q122)->q123
default34(qb, qb, qzeros)->q124
nonce(q124)->q125
default34(qb, qi, qzeros)->q126
nonce(q126)->q127
default34(qi, qa, qzeros)->q128
nonce(q128)->q129
default34(qi, qb, qzeros)->q130
nonce(q130)->q131
default34(qi, qi, qzeros)->q132
nonce(q132)->q133
sk(q29)->q134
text(qstart)->q135
sk(q17)->q138
sk(q13)->q140
```

Transitions

```
iknows(q49)->qnet
iknows(q55)->qnet
iknows(q57)->qnet
iknows(q59)->qnet
iknows(q61)->qnet
iknows(q67)->qnet
iknows(q73)->qnet
iknows(q75)->qnet
iknows(q77)->qnet
iknows(q79)->qnet
iknows(q85)->qnet
iknows(q91)->qnet
iknows(q93)->qnet
iknows(q95)->qnet
iknows(q97)->qnet
iknows(q103)->qnet
iknows(q109)->qnet
iknows(q111)->qnet
iknows(q113)->qnet
iknows(q115)->qnet
iknows(q121)->qnet
iknows(q127)->qnet
iknows(q129)->qnet
iknows(q131)->qnet
iknows(q133)->qnet
iknows(q135)->qnet
iknows(q135)->qnet
iknows(q135)->qnet
iknows(q135)->qnet
iknows(q135)->qnet
iknows(q135)->qnet
apply(qnet,qnet)->qnet
scrypt(qnet,qnet)->qnet
pair(qnet,qnet)->qnet
iknows(qnet)->qnet
iknows(q37)->qnet
iknows(qagt5)->qnet
iknows(qagt3)->qnet
iknows(q17)->qnet
iknows(q13)->qnet
iknows(q27)->qnet
iknows(qagti)->qnet
xor(qnet,qnet) -> qnet
state_alice(qagt5, qagt3, q134, q27, q43, q25, q25, q25, q25, q23, qnet)->qstate
state_bob(qagt3, qagt5, q134, q27, q43, q25, q25, q25, q21, q25, q19, qnet)->qstate
state_alice(qagt5, qagti, q138, q27, q43, q25, q25, q25, q25, q15, qnet)->qstate
state_alice(qagt3, qagti, q140, q27, q43, q25, q25, q25, q25, q11, qnet)->qstate
state_bob(qagt5, qagti, q138, q27, q43, q25, q25, q25, q21, q25, q9, qnet)->qstate
state_bob(qagt3, qagti, q140, q27, q43, q25, q25, q25, q21, q25, q7, qnet)->qstate
i->qi
```

```
agt(qi)->qagti
b->qb
agt(qb)->qagt3
a->qa
agt(qa)->qagt5
set_84->qset_84
msg(qset_84)->q7
set_81->qset_81
msg(qset_81)->q9
set_77->qset_77
msg(qset_77)->q11
kbi->qkbi
sk(qkbi)->q13
set_74->qset_74
msg(qset_74)->q15
kai->qkai
sk(qkai)->q17
set_69->qset_69
msg(qset_69)->q19
dummy_msg->qdummy_msg
msg(qdummy_msg)->q21
set_60->qset_60
msg(qset_60)->q23
dummy_nonce->qdummy_nonce
text(qdummy_nonce)->q25
h->qh
func(qh)->q27
kab->qkab
sk(qkab)->q29
nil->qnil
text(qnil)->q31
snb->qsnb
nat(qsnb)->q33
deux->qdeux
nat(qdeux)->q35
start->qstart
msg(qstart)->q37
sna->qsna
nat(qsna)->q39
un->qun
nat(qun)->q41
zeros->qzeros
nat(qzeros)->q43
default30(qa, qa, qzeros)->q44
nonce(q44)->q45
default30(qa, qb, qzeros)->q46
nonce(q46)->q47
default30(qa, qi, qzeros)->q48
nonce(q48)->q49
default30(qb, qa, qzeros)->q50
nonce(q50)->q51
default30(qb, qb, qzeros)->q52
```

```
nonce(q52)->q53
default30(qb, qi, qzeros)->q54
nonce(q54)->q55
default30(qi, qa, qzeros)->q56
nonce(q56)->q57
default30(qi, qb, qzeros)->q58
nonce(q58)->q59
default30(qi, qi, qzeros)->q60
nonce(q60)->q61
default31(qa, qa, qzeros)->q62
nonce(q62)->q63
default31(qa, qb, qzeros)->q64
nonce(q64)->q65
default31(qa, qi, qzeros)->q66
nonce(q66)->q67
default31(qb, qa, qzeros)->q68
nonce(q68)->q69
default31(qb, qb, qzeros)->q70
nonce(q70)->q71
default31(qb, qi, qzeros)->q72
nonce(q72)->q73
default31(qi, qa, qzeros)->q74
nonce(q74)->q75
default31(qi, qb, qzeros)->q76
nonce(q76)->q77
default31(qi, qi, qzeros)->q78
nonce(q78)->q79
default32(qa, qa, qzeros)->q80
nonce(q80)->q81
default32(qa, qb, qzeros)->q82
nonce(q82)->q83
default32(qa, qi, qzeros)->q84
nonce(q84)->q85
default32(qb, qa, qzeros)->q86
nonce(q86)->q87
default32(qb, qb, qzeros)->q88
nonce(q88)->q89
default32(qb, qi, qzeros)->q90
nonce(q90)->q91
default32(qi, qa, qzeros)->q92
nonce(q92)->q93
default32(qi, qb, qzeros)->q94
nonce(q94)->q95
default32(qi, qi, qzeros)->q96
nonce(q96)->q97
default33(qa, qa, qzeros)->q98
nonce(q98)->q99
default33(qa, qb, qzeros)->q100
nonce(q100)->q101
default33(qa, qi, qzeros)->q102
nonce(q102)->q103
default33(qb, qa, qzeros)->q104
```

```
nonce(q104)->q105
default33(qb, qb, qzeros)->q106
nonce(q106)->q107
default33(qb, qi, qzeros)->q108
nonce(q108)->q109
default33(qi, qa, qzeros)->q110
nonce(q110)->q111
default33(qi, qb, qzeros)->q112
nonce(q112)->q113
default33(qi, qi, qzeros)->q114
nonce(q114)->q115
default34(qa, qa, qzeros)->q116
nonce(q116)->q117
default34(qa, qb, qzeros)->q118
nonce(q118)->q119
default34(qa, qi, qzeros)->q120
nonce(q120)->q121
default34(qb, qa, qzeros)->q122
nonce(q122)->q123
default34(qb, qb, qzeros)->q124
nonce(q124)->q125
default34(qb, qi, qzeros)->q126
nonce(q126)->q127
default34(qi, qa, qzeros)->q128
nonce(q128)->q129
default34(qi, qb, qzeros)->q130
nonce(q130)->q131
default34(qi, qi, qzeros)->q132
nonce(q132)->q133
sk(q29)->q134
text(qstart)->q135
sk(q17)->q138
sk(q13)->q140



Approximation symbolic

States qapprox[0--60] qnet qstate qagta qagti

Rules

[state_alice(x49, x48, x47, x46, x45, x44, x43, x42, x41, SSetstate_alice,
iknows(scrypt(x40,pair(x39,xor(x38,x37))))) -> delphine]
->[xor(x38,x37)->x50 pair(x39,x50)->x51 scrypt(x40,x51)->x52
iknows(x52)->qnet state_alice(x49, x48, x47, x46, x45, x44, x43, x42, x41,
SSetstate_alice, qnet)->delphine]

[state_bob(x36, x35, x34, x33, x32, x31, x30, x29, xor(x28,x27), x26,
SSetstate_bob, iknows(x25)) -> delphine] ->[iknows(x25)->qnet
xor(x28,x27)->x56 state_bob(x36, x35, x34, x33, x32, x31, x30, x29, x56,
x26, SSetstate_bob, qnet)->delphine]
```

```
[state_alice(x24, x23, x22, x21, x20, x19, x18, x17, x16, SSetstate_alice,
iknows(pair(x15,scrypt(x14,apply(x13,x12))))) -> delphine]
->[apply(x13,x12)->x58 scrypt(x14,x58)->x59 pair(x15,x59)->x60
iknows(x60)->qnet state_alice(x24, x23, x22, x21, x20, x19, x18, x17, x16,
SSetstate_alice, qnet)->delphine]

[state_bob(x12, x11, x10, x9, x8, x7, x6, x5, xor(x4,xor(x3,x2)), x1,
SSetstate_bob, iknows(x0)) -> delphine] ->[iknows(x0)->qnet xor(x3,x2)->x64
xor(x4,x64)->x65 state_bob(x12, x11, x10, x9, x8, x7, x6, x5, x65, x1,
SSetstate_bob, qnet)->delphine]

[z -> z] ->[]

[z -> z] ->[]

[x -> x] ->[]

[y -> y] ->[]

[x -> x] ->[]


[e -> x] -> [e -> x]

[ xor(y,x) -> delphine] -> [ xor(y,x) -> delphine]

[xor(xor(x,y),z) -> delphine] -> [xor(x,y)-> x1 xor(x1,z)-> delphine]

[x -> x] -> []




PropertiesDeclaration [0:[secret0sna(nonce(default31(Astate_alice,
Bstate_alice,
zeros)),sna,SSetstate_alice,agt(Bstate_alice),agt(Astate_alice)) ]]
[3:[secret3snb(nonce(Xdefault31),snb,SSetstate_bob,agt(Bstate_bob),agt(Astate_bob))
]]


NonLinear
[10:[[x y]]]
```

5.1 Timbuk Input Format of Secret Terms

```
Automaton etatInitial

States qa qb qzeros qdata qnonce qf

Final States qf
```

```
Transitions

a -> qa
b -> qb
zeros -> qzeros
default31(qa,qa,qzeros) -> qdata
default31(qb,qa,qzeros) -> qdata
default31(qa,qb,qzeros) -> qdata
default31(qb,qb,qzeros) -> qdata
nonce(qdata) -> qnonce
iknows(qnonce) -> qf
```

Timbuk Input Format of the fixed Point Automaton

```
States qapprox0:0 qapprox1:0 qapprox2:0 qapprox3:0 qapprox4:0 qapprox5:0
 qapprox6:0 qapprox7:0 qapprox8:0 qapprox9:0 qapprox10:0 qapprox11:0
 qapprox12:0 qapprox13:0 qapprox14:0 qapprox15:0 qapprox16:0 qapprox17:0
 qapprox18:0 qapprox19:0 qapprox20:0 qapprox21:0 qapprox22:0 qapprox23:0
 qapprox24:0 qapprox25:0 qapprox26:0 qapprox27:0 qapprox28:0 qapprox29:0
 qapprox30:0 qapprox31:0 qapprox32:0 qapprox33:0 qapprox34:0 qapprox35:0
 qapprox36:0 qapprox37:0 qapprox38:0 qapprox39:0 qapprox40:0 qapprox41:0
 qapprox42:0 qapprox43:0 qapprox44:0 qapprox45:0 qapprox46:0 qapprox47:0
 qapprox48:0 qapprox49:0 qapprox50:0 qapprox51:0 qapprox52:0 qapprox53:0
 qapprox54:0 qapprox55:0 qapprox56:0 qapprox57:0 qapprox58:0 qapprox59:0
 qapprox60:0 qagta:0 qi:0 qagti:0 qb:0 qagt3:0 qa:0 qagt5:0 qset_84:0 q7:0
 qset_81:0 q9:0 qset_77:0 q11:0 qkbi:0 q13:0 qset_74:0 q15:0 qkai:0 q17:0
 qset_69:0 q19:0 qdummy_msg:0 q21:0 qset_60:0 q23:0 qdummy_nonce:0 q25:0
 qh:0 q27:0 qkab:0 q29:0 qnil:0 q31:0 qsnb:0 q33:0 qdeu x:0 q35:0 qstart:0
 q37:0 qsna:0 q39:0 qun:0 q41:0 qzeros:0 q43:0 q44:0 q45:0 q46:0 q47:0 q48:0
 q49:0 q50:0 q51:0 q52:0 q53:0 q54:0 q55:0 q56:0 q57:0 q58:0 q59:0 q60:0
 q61:0 q62:0 q63:0 q64:0 q65:0 q66:0 q67:0 q68:0 q69:0 q70:0 q71:0 q72:0
 q73:0 q74:0 q75:0 q76:0 q77:0 q78:0 q79:0 q80:0 q81:0 q82:0 q83:0 q84:0
 q85:0 q86:0 q87:0 q88:0 q89:0 q90:0 q91:0 q92:0 q93:0 q94:0 q95:0 q96:0
 q97:0 q98:0 q99:0 q100:0 q101:0 q102:0 q103:0 q104:0 q105:0 q106:0 q107:0
 q108:0 q109:0 q110:0 q111:0 q112:0 q113:0 q114:0 q115:0 q116:0 q117:0
 q118:0 q119:0 q120:0 q121:0 q122:0 q123:0 q124:0 q125:0 q126:0 q127:0
 q128:0 q129:0 q130:0 q131:0 q132:0 q133:0 q134:0 q135:0 q138:0 q140:0
 qstate:0 qnet:0

Final States qnet qstate

Prior
i -> qi
agt(qi) -> qagti
b -> qb
agt(qb) -> qagt3
a -> qa
```

```
agt(qa) -> qagt5
set_84 -> qset_84
msg(qset_84) -> q7
set_81 -> qset_81
msg(qset_81) -> q9
set_77 -> qset_77
msg(qset_77) -> q11
kbi -> qkbi
sk(qkbi) -> q13
set_74 -> qset_74
msg(qset_74) -> q15
kai -> qkai
sk(qkai) -> q17
set_69 -> qset_69
msg(qset_69) -> q19
dummy_msg -> qdummy_msg
msg(qdummy_msg) -> q21
set_60 -> qset_60
msg(qset_60) -> q23
dummy_nonce -> qdummy_nonce
text(qdummy_nonce) -> q25
h -> qh
func(qh) -> q27
kab -> qkab
sk(qkab) -> q29
nil -> qnil
text(qnil) -> q31
snb -> qsnb
nat(qsnb) -> q33
deux -> qdeux
nat(qdeux) -> q35
start -> qstart
msg(qstart) -> q37
sna -> qsna
nat(qsna) -> q39
un -> qun
nat(qun) -> q41
zeros -> qzeros
nat(qzeros) -> q43
default30(qa,qa,qzeros) -> q44
nonce(q44) -> q45
default30(qa,qb,qzeros) -> q46
nonce(q46) -> q47
default30(qa,qi,qzeros) -> q48
nonce(q48) -> q49
default30(qb,qa,qzeros) -> q50
nonce(q50) -> q51
default30(qb,qb,qzeros) -> q52
nonce(q52) -> q53
default30(qb,qi,qzeros) -> q54
nonce(q54) -> q55
default30(qi,qa,qzeros) -> q56
```

```
nonce(q56) -> q57
default30(qi,qb,qzeros) -> q58
nonce(q58) -> q59
default30(qi,qi,qzeros) -> q60
nonce(q60) -> q61
default31(qa,qa,qzeros) -> q62
nonce(q62) -> q63
default31(qa,qb,qzeros) -> q64
nonce(q64) -> q65
default31(qa,qi,qzeros) -> q66
nonce(q66) -> q67
default31(qb,qa,qzeros) -> q68
nonce(q68) -> q69
default31(qb,qb,qzeros) -> q70
nonce(q70) -> q71
default31(qb,qi,qzeros) -> q72
nonce(q72) -> q73
default31(qi,qa,qzeros) -> q74
nonce(q74) -> q75
default31(qi,qb,qzeros) -> q76
nonce(q76) -> q77
default31(qi,qi,qzeros) -> q78
nonce(q78) -> q79
default32(qa,qa,qzeros) -> q80
nonce(q80) -> q81
default32(qa,qb,qzeros) -> q82
nonce(q82) -> q83
default32(qa,qi,qzeros) -> q84
nonce(q84) -> q85
default32(qb,qa,qzeros) -> q86
nonce(q86) -> q87
default32(qb,qb,qzeros) -> q88
nonce(q88) -> q89
default32(qb,qi,qzeros) -> q90
nonce(q90) -> q91
default32(qi,qa,qzeros) -> q92
nonce(q92) -> q93
default32(qi,qb,qzeros) -> q94
nonce(q94) -> q95
default32(qi,qi,qzeros) -> q96
nonce(q96) -> q97
default33(qa,qa,qzeros) -> q98
nonce(q98) -> q99
default33(qa,qb,qzeros) -> q100
nonce(q100) -> q101
default33(qa,qi,qzeros) -> q102
nonce(q102) -> q103
default33(qb,qa,qzeros) -> q104
nonce(q104) -> q105
default33(qb,qb,qzeros) -> q106
nonce(q106) -> q107
default33(qb,qi,qzeros) -> q108
```

```
nonce(q108) -> q109
default33(qi,qa,qzeros) -> q110
nonce(q110) -> q111
default33(qi,qb,qzeros) -> q112
nonce(q112) -> q113
default33(qi,qi,qzeros) -> q114
nonce(q114) -> q115
default34(qa,qa,qzeros) -> q116
nonce(q116) -> q117
default34(qa,qb,qzeros) -> q118
nonce(q118) -> q119
default34(qa,qi,qzeros) -> q120
nonce(q120) -> q121
default34(qb,qa,qzeros) -> q122
nonce(q122) -> q123
default34(qb,qb,qzeros) -> q124
nonce(q124) -> q125
default34(qb,qi,qzeros) -> q126
nonce(q126) -> q127
default34(qi,qa,qzeros) -> q128
nonce(q128) -> q129
default34(qi,qb,qzeros) -> q130
nonce(q130) -> q131
default34(qi,qi,qzeros) -> q132
nonce(q132) -> q133
sk(q29) -> q134
text(qstart) -> q135
sk(q17) -> q138
sk(q13) -> q140

Transitions
iknows(q49) -> qnet
iknows(q55) -> qnet
iknows(q57) -> qnet
iknows(q59) -> qnet
iknows(q61) -> qnet
iknows(q67) -> qnet
iknows(q73) -> qnet
iknows(q75) -> qnet
iknows(q77) -> qnet
iknows(q79) -> qnet
iknows(q85) -> qnet
iknows(q91) -> qnet
iknows(q93) -> qnet
iknows(q95) -> qnet
iknows(q97) -> qnet
iknows(q103) -> qnet
iknows(q109) -> qnet
iknows(q111) -> qnet
iknows(q113) -> qnet
iknows(q115) -> qnet
iknows(q121) -> qnet
```

```
iknows(q127) -> qnet
iknows(q129) -> qnet
iknows(q131) -> qnet
iknows(q133) -> qnet
iknows(q135) -> qnet
apply(qnet,qnet) -> qnet
scrypt(qnet,qnet) -> qnet
pair(qnet,qnet) -> qnet
iknows(qnet) -> qnet
iknows(q37) -> qnet
iknows(qagt5) -> qnet
iknows(qagt3) -> qnet
iknows(q17) -> qnet
iknows(q13) -> qnet
iknows(q27) -> qnet
iknows(qagti) -> qnet
xor(qnet,qnet) -> qnet
state_alice(qagt5,qagt3,q134,q27,q43,q25,q25,q25,q25,q23,qnet) -> qstate
state_bob(qagt3,qagt5,q134,q27,q43,q25,q25,q25,q21,q25,q19,qnet) -> qstate
state_alice(qagt5,qagti,q138,q27,q43,q25,q25,q25,q25,q15,qnet) -> qstate
state_alice(qagt3,qagti,q140,q27,q43,q25,q25,q25,q25,q11,qnet) -> qstate
state_bob(qagt5,qagti,q138,q27,q43,q25,q25,q25,q21,q25,q9,qnet) -> qstate
state_bob(qagt3,qagti,q140,q27,q43,q25,q25,q25,q21,q25,q7,qnet) -> qstate
i -> qi
agt(qi) -> qagti
b -> qb
agt(qb) -> qagt3
a -> qa
agt(qa) -> qagt5
set_84 -> qset_84
msg(qset_84) -> q7
set_81 -> qset_81
msg(qset_81) -> q9
set_77 -> qset_77
msg(qset_77) -> q11
kbi -> qkbi
sk(qkbi) -> q13
set_74 -> qset_74
msg(qset_74) -> q15
kai -> qkai
sk(qkai) -> q17
set_69 -> qset_69
msg(qset_69) -> q19
dummy_msg -> qdummy_msg
msg(qdummy_msg) -> q21
set_60 -> qset_60
msg(qset_60) -> q23
dummy_nonce -> qdummy_nonce
text(qdummy_nonce) -> q25
h -> qh
func(qh) -> q27
kab -> qkab
```

```
sk(qkab) -> q29
nil -> qnil
text(qnil) -> q31
snb -> qsnb
nat(qsnb) -> q33
deux -> qdeux
nat(qdeux) -> q35
start -> qstart
msg(qstart) -> q37
sna -> qsna
nat(qsna) -> q39
un -> qun
nat(qun) -> q41
zeros -> qzeros
nat(qzeros) -> q43
default30(qa,qa,qzeros) -> q44
nonce(q44) -> q45
default30(qa,qb,qzeros) -> q46
nonce(q46) -> q47
default30(qa,qi,qzeros) -> q48
nonce(q48) -> q49
default30(qb,qa,qzeros) -> q50
nonce(q50) -> q51
default30(qb,qb,qzeros) -> q52
nonce(q52) -> q53
default30(qb,qi,qzeros) -> q54
nonce(q54) -> q55
default30(qi,qa,qzeros) -> q56
nonce(q56) -> q57
default30(qi,qb,qzeros) -> q58
nonce(q58) -> q59
default30(qi,qi,qzeros) -> q60
nonce(q60) -> q61
default31(qa,qa,qzeros) -> q62
nonce(q62) -> q63
default31(qa,qb,qzeros) -> q64
nonce(q64) -> q65
default31(qa,qi,qzeros) -> q66
nonce(q66) -> q67
default31(qb,qa,qzeros) -> q68
nonce(q68) -> q69
default31(qb,qb,qzeros) -> q70
nonce(q70) -> q71
default31(qb,qi,qzeros) -> q72
nonce(q72) -> q73
default31(qi,qa,qzeros) -> q74
nonce(q74) -> q75
default31(qi,qb,qzeros) -> q76
nonce(q76) -> q77
default31(qi,qi,qzeros) -> q78
nonce(q78) -> q79
default32(qa,qa,qzeros) -> q80
```

```
nonce(q80) -> q81
default32(qa,qb,qzeros) -> q82
nonce(q82) -> q83
default32(qa,qi,qzeros) -> q84
nonce(q84) -> q85
default32(qb,qa,qzeros) -> q86
nonce(q86) -> q87
default32(qb,qb,qzeros) -> q88
nonce(q88) -> q89
default32(qb,qi,qzeros) -> q90
nonce(q90) -> q91
default32(qi,qa,qzeros) -> q92
nonce(q92) -> q93
default32(qi,qb,qzeros) -> q94
nonce(q94) -> q95
default32(qi,qi,qzeros) -> q96
nonce(q96) -> q97
default33(qa,qa,qzeros) -> q98
nonce(q98) -> q99
default33(qa,qb,qzeros) -> q100
nonce(q100) -> q101
default33(qa,qi,qzeros) -> q102
nonce(q102) -> q103
default33(qb,qa,qzeros) -> q104
nonce(q104) -> q105
default33(qb,qb,qzeros) -> q106
nonce(q106) -> q107
default33(qb,qi,qzeros) -> q108
nonce(q108) -> q109
default33(qi,qa,qzeros) -> q110
nonce(q110) -> q111
default33(qi,qb,qzeros) -> q112
nonce(q112) -> q113
default33(qi,qi,qzeros) -> q114
nonce(q114) -> q115
default34(qa,qa,qzeros) -> q116
nonce(q116) -> q117
default34(qa,qb,qzeros) -> q118
nonce(q118) -> q119
default34(qa,qi,qzeros) -> q120
nonce(q120) -> q121
default34(qb,qa,qzeros) -> q122
nonce(q122) -> q123
default34(qb,qb,qzeros) -> q124
nonce(q124) -> q125
default34(qb,qi,qzeros) -> q126
nonce(q126) -> q127
default34(qi,qa,qzeros) -> q128
nonce(q128) -> q129
default34(qi,qb,qzeros) -> q130
nonce(q130) -> q131
default34(qi,qi,qzeros) -> q132
```

```
nonce(q132) -> q133
sk(q29) -> q134
text(qstart) -> q135
sk(q17) -> q138
sk(q13) -> q140
state_alice(qagt5,qagt3,q134,q27,q41,q47,q83,q65,q25,q23,qnet) -> qstate
iknows(qapprox17) -> qnet
scrypt(q134,qapprox16) -> qapprox17
pair(q47,qapprox15) -> qapprox16
xor(q65,q83) -> qapprox15
state_alice(qagt3,qagti,q140,q27,q41,q55,q91,q73,q25,q11,qnet) -> qstate
scrypt(q140,qapprox11) -> qapprox12
pair(q55,qapprox10) -> qapprox11
xor(q73,q91) -> qapprox10
state_alice(qagt5,qagti,q138,q27,q41,q49,q85,q67,q25,q15,qnet) -> qstate
iknows(qapprox12) -> qnet
scrypt(q138,qapprox11) -> qapprox12
pair(q49,qapprox10) -> qapprox11
xor(q67,q85) -> qapprox10
nonce(q48) -> qnet
nonce(q56) -> qnet
nonce(q60) -> qnet
nonce(q72) -> qnet
nonce(q76) -> qnet
nonce(q84) -> qnet
nonce(q92) -> qnet
nonce(q96) -> qnet
nonce(q108) -> qnet
nonce(q112) -> qnet
nonce(q120) -> qnet
nonce(q128) -> qnet
nonce(q132) -> qnet
agt(qa) -> qnet
sk(qkai) -> qnet
func(qh) -> qnet
agt(qi) -> qnet
sk(qkbi) -> qnet
agt(qb) -> qnet
msg(qstart) -> qnet
text(qstart) -> qnet
nonce(q130) -> qnet
nonce(q126) -> qnet
nonce(q114) -> qnet
nonce(q110) -> qnet
nonce(q102) -> qnet
nonce(q94) -> qnet
nonce(q90) -> qnet
nonce(q78) -> qnet
nonce(q74) -> qnet
nonce(q66) -> qnet
nonce(q58) -> qnet
nonce(q54) -> qnet
```

```
e -> qnet
e -> qapprox16
e -> qapprox11
state_alice(qagt3,qagti,q140,q27,q35,q55,q91,q73,q61,q11,qnet) -> qstate
state_alice(qagt3,qagti,q140,q27,q35,q55,q91,q73,q93,q11,qnet) -> qstate
state_alice(qagt3,qagti,q140,q27,q35,q55,q91,q73,q121,q11,qnet) -> qstate
state_alice(qagt3,qagti,q140,q27,q35,q55,q91,q73,q127,q11,qnet) -> qstate
state_alice(qagt3,qagti,q140,q27,q35,q55,q91,q73,q95,q11,qnet) -> qstate
state_alice(qagt3,qagti,q140,q27,q35,q55,q91,q73,q67,q11,qnet) -> qstate
state_alice(qagt3,qagti,q140,q27,q35,q55,q91,q73,q59,q11,qnet) -> qstate
state_alice(qagt3,qagti,q140,q27,q35,q55,q91,q73,q91,q11,qnet) -> qstate
state_alice(qagt3,qagti,q140,q27,q35,q55,q91,q73,q115,q11,qnet) -> qstate
state_alice(qagt3,qagti,q140,q27,q35,q55,q91,q73,q131,q11,qnet) -> qstate
state_alice(qagt3,qagti,q140,q27,q35,q55,q91,q73,q103,q11,qnet) -> qstate
state_alice(qagt3,qagti,q140,q27,q35,q55,q91,q73,q75,q11,qnet) -> qstate
state_alice(qagt3,qagti,q140,q27,q35,q55,q91,q73,q55,q11,qnet) -> qstate
state_alice(qagt3,qagti,q140,q27,q35,q55,q91,q73,q79,q11,qnet) -> qstate
state_alice(qagt3,qagti,q140,q27,q35,q55,q91,q73,q111,q11,qnet) -> qstate
state_alice(qagt3,qagti,q140,q27,q35,q55,q91,q73,q133,q11,qnet) -> qstate
state_alice(qagt3,qagti,q140,q27,q35,q55,q91,q73,q109,q11,qnet) -> qstate
state_alice(qagt3,qagti,q140,q27,q35,q55,q91,q73,q77,q11,qnet) -> qstate
state_alice(qagt3,qagti,q140,q27,q35,q55,q91,q73,q49,q11,qnet) -> qstate
state_alice(qagt3,qagti,q140,q27,q35,q55,q91,q73,q129,q11,qnet) -> qstate
state_alice(qagt3,qagti,q140,q27,q35,q55,q91,q73,q113,q11,qnet) -> qstate
state_alice(qagt3,qagti,q140,q27,q35,q55,q91,q73,q97,q11,qnet) -> qstate
state_alice(qagt3,qagti,q140,q27,q35,q55,q91,q73,q85,q11,qnet) -> qstate
state_alice(qagt3,qagti,q140,q27,q35,q55,q91,q73,q73,q11,qnet) -> qstate
state_alice(qagt3,qagti,q140,q27,q35,q55,q91,q73,q57,q11,qnet) -> qstate
pair(q91,qapprox59) -> qapprox60
scrypt(q55,qapprox63) -> qapprox59
state_alice(qagt5,qagti,q138,q27,q35,q49,q85,q67,q61,q15,qnet) -> qstate
state_alice(qagt5,qagti,q138,q27,q35,q49,q85,q67,q93,q15,qnet) -> qstate
state_alice(qagt5,qagti,q138,q27,q35,q49,q85,q67,q121,q15,qnet) -> qstate
state_alice(qagt5,qagti,q138,q27,q35,q49,q85,q67,q127,q15,qnet) -> qstate
state_alice(qagt5,qagti,q138,q27,q35,q49,q85,q67,q95,q15,qnet) -> qstate
state_alice(qagt5,qagti,q138,q27,q35,q49,q85,q67,q67,q15,qnet) -> qstate
state_alice(qagt5,qagti,q138,q27,q35,q49,q85,q67,q59,q15,qnet) -> qstate
state_alice(qagt5,qagti,q138,q27,q35,q49,q85,q67,q91,q15,qnet) -> qstate
state_alice(qagt5,qagti,q138,q27,q35,q49,q85,q67,q115,q15,qnet) -> qstate
state_alice(qagt5,qagti,q138,q27,q35,q49,q85,q67,q131,q15,qnet) -> qstate
state_alice(qagt5,qagti,q138,q27,q35,q49,q85,q67,q103,q15,qnet) -> qstate
state_alice(qagt5,qagti,q138,q27,q35,q49,q85,q67,q75,q15,qnet) -> qstate
state_alice(qagt5,qagti,q138,q27,q35,q49,q85,q67,q55,q15,qnet) -> qstate
state_alice(qagt5,qagti,q138,q27,q35,q49,q85,q67,q79,q15,qnet) -> qstate
state_alice(qagt5,qagti,q138,q27,q35,q49,q85,q67,q111,q15,qnet) -> qstate
state_alice(qagt5,qagti,q138,q27,q35,q49,q85,q67,q133,q15,qnet) -> qstate
state_alice(qagt5,qagti,q138,q27,q35,q49,q85,q67,q109,q15,qnet) -> qstate
state_alice(qagt5,qagti,q138,q27,q35,q49,q85,q67,q77,q15,qnet) -> qstate
state_alice(qagt5,qagti,q138,q27,q35,q49,q85,q67,q49,q15,qnet) -> qstate
state_alice(qagt5,qagti,q138,q27,q35,q49,q85,q67,q129,q15,qnet) -> qstate
state_alice(qagt5,qagti,q138,q27,q35,q49,q85,q67,q113,q15,qnet) -> qstate
state_alice(qagt5,qagti,q138,q27,q35,q49,q85,q67,q97,q15,qnet) -> qstate
```

```
state_alice(qagt5,qagti,q138,q27,q35,q49,q85,q67,q85,q15,qnet) -> qstate
state_alice(qagt5,qagti,q138,q27,q35,q49,q85,q67,q73,q15,qnet) -> qstate
state_alice(qagt5,qagti,q138,q27,q35,q49,q85,q67,q57,q15,qnet) -> qstate
iknows(qapprox60) -> qnet
pair(q85,qapprox59) -> qapprox60
scrypt(q49,qapprox63) -> qapprox59
state_alice(qagt5,qagt3,q134,q27,q35,q47,q83,q65,q61,q23,qnet) -> qstate
apply(q27,q61) -> qapprox63
state_alice(qagt5,qagt3,q134,q27,q35,q47,q83,q65,q93,q23,qnet) -> qstate
apply(q27,q93) -> qapprox63
state_alice(qagt5,qagt3,q134,q27,q35,q47,q83,q65,q121,q23,qnet) -> qstate
apply(q27,q121) -> qapprox63
state_alice(qagt5,qagt3,q134,q27,q35,q47,q83,q65,q127,q23,qnet) -> qstate
apply(q27,q127) -> qapprox63
state_alice(qagt5,qagt3,q134,q27,q35,q47,q83,q65,q95,q23,qnet) -> qstate
apply(q27,q95) -> qapprox63
state_alice(qagt5,qagt3,q134,q27,q35,q47,q83,q65,q67,q23,qnet) -> qstate
apply(q27,q67) -> qapprox63
state_alice(qagt5,qagt3,q134,q27,q35,q47,q83,q65,q59,q23,qnet) -> qstate
apply(q27,q59) -> qapprox63
state_alice(qagt5,qagt3,q134,q27,q35,q47,q83,q65,q91,q23,qnet) -> qstate
apply(q27,q91) -> qapprox63
state_alice(qagt5,qagt3,q134,q27,q35,q47,q83,q65,q115,q23,qnet) -> qstate
apply(q27,q115) -> qapprox63
state_alice(qagt5,qagt3,q134,q27,q35,q47,q83,q65,q131,q23,qnet) -> qstate
apply(q27,q131) -> qapprox63
state_alice(qagt5,qagt3,q134,q27,q35,q47,q83,q65,q103,q23,qnet) -> qstate
apply(q27,q103) -> qapprox63
state_alice(qagt5,qagt3,q134,q27,q35,q47,q83,q65,q75,q23,qnet) -> qstate
apply(q27,q75) -> qapprox63
state_alice(qagt5,qagt3,q134,q27,q35,q47,q83,q65,q55,q23,qnet) -> qstate
apply(q27,q55) -> qapprox63
state_alice(qagt5,qagt3,q134,q27,q35,q47,q83,q65,q79,q23,qnet) -> qstate
apply(q27,q79) -> qapprox63
state_alice(qagt5,qagt3,q134,q27,q35,q47,q83,q65,q111,q23,qnet) -> qstate
apply(q27,q111) -> qapprox63
state_alice(qagt5,qagt3,q134,q27,q35,q47,q83,q65,q133,q23,qnet) -> qstate
apply(q27,q133) -> qapprox63
state_alice(qagt5,qagt3,q134,q27,q35,q47,q83,q65,q109,q23,qnet) -> qstate
apply(q27,q109) -> qapprox63
state_alice(qagt5,qagt3,q134,q27,q35,q47,q83,q65,q77,q23,qnet) -> qstate
apply(q27,q77) -> qapprox63
state_alice(qagt5,qagt3,q134,q27,q35,q47,q83,q65,q49,q23,qnet) -> qstate
apply(q27,q49) -> qapprox63
state_alice(qagt5,qagt3,q134,q27,q35,q47,q83,q65,q129,q23,qnet) -> qstate
apply(q27,q129) -> qapprox63
state_alice(qagt5,qagt3,q134,q27,q35,q47,q83,q65,q113,q23,qnet) -> qstate
apply(q27,q113) -> qapprox63
state_alice(qagt5,qagt3,q134,q27,q35,q47,q83,q65,q97,q23,qnet) -> qstate
apply(q27,q97) -> qapprox63
state_alice(qagt5,qagt3,q134,q27,q35,q47,q83,q65,q85,q23,qnet) -> qstate
apply(q27,q85) -> qapprox63
```

```
state_alice(qagt5,qagt3,q134,q27,q35,q47,q83,q65,q73,q23,qnet) -> qstate
apply(q27,q73) -> qapprox63
state_alice(qagt5,qagt3,q134,q27,q35,q47,q83,q65,q57,q23,qnet) -> qstate
iknows(qapprox65) -> qnet
pair(q83,qapprox64) -> qapprox65
scrypt(q47,qapprox63) -> qapprox64
apply(q27,q57) -> qapprox63
nonce(q46) -> qapprox16
xor(q65,q83) -> qapprox16
nonce(q54) -> qapprox11
nonce(q48) -> qapprox11
xor(q73,q91) -> qapprox11
xor(q67,q85) -> qapprox11
scrypt(q134,qapprox16) -> qnet
scrypt(q140,qapprox11) -> qnet
scrypt(q138,qapprox11) -> qnet
e -> qapprox15
e -> qapprox10
state_bob(qagt3,qagt5,q134,q27,q41,q47,q25,q25,qapprox15,q123,q19,qnet) -> qstate
iknows(q123) -> qnet
state_bob(qagt3,qagti,q140,q27,q41,q55,q25,q25,qapprox10,q127,q7,qnet) -> qstate
state_bob(qagt3,qagti,q140,q27,q41,q49,q25,q25,qapprox10,q127,q7,qnet) -> qstate
state_bob(qagt5,qagti,q138,q27,q41,q55,q25,q25,qapprox10,q121,q9,qnet) -> qstate
state_bob(qagt5,qagti,q138,q27,q41,q49,q25,q25,qapprox10,q121,q9,qnet) -> qstate
scrypt(q49,qapprox58) -> qapprox60
scrypt(q55,qapprox58) -> qapprox60
scrypt(q47,qapprox58) -> qapprox65
scrypt(q55,qapprox63) -> qnet
scrypt(q49,qapprox63) -> qnet
nonce(q82) -> qnet
scrypt(q47,qapprox63) -> qnet
state_alice(qagt5,qagt3,q134,q27,q35,q47,q83,q65,q123,q23,qnet) -> qstate
scrypt(q47,qapprox58) -> qapprox64
state_alice(qagt5,qagti,q138,q27,q35,q49,q85,q67,q123,q15,qnet) -> qstate
scrypt(q49,qapprox58) -> qapprox59
state_alice(qagt3,qagti,q140,q27,q35,q55,q91,q73,q123,q11,qnet) -> qstate
scrypt(q55,qapprox58) -> qapprox59
apply(q27,q123) -> qapprox58
state_bob(qagt3,qagti,q140,q27,q35,q55,q85,q67,qapprox88,q127,q7,qnet) -> qstate
state_bob(qagt3,qagti,q140,q27,q35,q55,q91,q73,qapprox88,q127,q7,qnet) -> qstate
state_bob(qagt5,qagti,q138,q27,q35,q55,q85,q67,qapprox88,q121,q9,qnet) -> qstate
state_bob(qagt5,qagti,q138,q27,q35,q55,q91,q73,qapprox88,q121,q9,qnet) -> qstate
state_bob(qagt5,qagti,q138,q27,q35,q49,q85,q67,qapprox88,q121,q9,qnet) -> qstate
state_bob(qagt5,qagti,q138,q27,q35,q49,q91,q73,qapprox88,q121,q9,qnet) -> qstate
state_bob(qagt3,qagti,q140,q27,q35,q49,q85,q67,qapprox88,q127,q7,qnet) -> qstate
xor(q85,qapprox10) -> qapprox88
state_bob(qagt3,qagti,q140,q27,q35,q49,q91,q73,qapprox88,q127,q7,qnet) -> qstate
xor(q91,qapprox10) -> qapprox88
iknows(q31) -> qnet
nonce(q90) -> qapprox60
nonce(q84) -> qapprox60
scrypt(q55,qapprox63) -> qapprox60
```

```
scrypt(q49,qapprox63) -> qapprox60
nonce(q82) -> qapprox65
scrypt(q47,qapprox63) -> qapprox65
pair(q91,qapprox59) -> qnet
pair(q85,qapprox59) -> qnet
nonce(q122) -> qnet
pair(q83,qapprox64) -> qnet
apply(q27,q83) -> qnet
state_alice(qagt3,qagti,q140,q27,q35,q55,q91,q73,q83,q11,qnet) -> qstate
state_alice(qagt5,qagti,q138,q27,q35,q49,q85,q67,q83,q15,qnet) -> qstate
state_alice(qagt5,qagt3,q134,q27,q35,q47,q83,q65,q83,q23,qnet) -> qstate
apply(q27,q83) -> qapprox63
state_bob(qagt3,qagt5,q134,q27,q35,q47,q83,q65,qapprox92,q123,q19,qnet) -> qstate
xor(q83,qapprox15) -> qapprox92
apply(q27,q61) -> qnet
apply(q27,q93) -> qnet
apply(q27,q121) -> qnet
apply(q27,q127) -> qnet
apply(q27,q95) -> qnet
apply(q27,q67) -> qnet
apply(q27,q59) -> qnet
apply(q27,q91) -> qnet
apply(q27,q115) -> qnet
apply(q27,q131) -> qnet
apply(q27,q103) -> qnet
apply(q27,q75) -> qnet
apply(q27,q55) -> qnet
apply(q27,q79) -> qnet
apply(q27,q111) -> qnet
apply(q27,q133) -> qnet
apply(q27,q109) -> qnet
apply(q27,q77) -> qnet
apply(q27,q49) -> qnet
apply(q27,q129) -> qnet
apply(q27,q113) -> qnet
apply(q27,q97) -> qnet
apply(q27,q85) -> qnet
apply(q27,q73) -> qnet
apply(q27,q57) -> qnet
scrypt(q49,qapprox58) -> qnet
scrypt(q55,qapprox58) -> qnet
scrypt(q47,qapprox58) -> qnet
text(qnil) -> qnet
e -> qapprox88
xor(q85,q73) -> qapprox114
xor(q85,q67) -> qapprox114
xor(qapprox114,q91) -> qapprox88
xor(q91,q73) -> qapprox114
xor(qapprox114,q85) -> qapprox88
xor(q91,q67) -> qapprox114
nonce(q84) -> qapprox88
nonce(q90) -> qapprox88
```

```
apply(q27,q123) -> qnet
e -> qapprox92
xor(qapprox114,q83) -> qapprox92
xor(q83,q65) -> qapprox114
nonce(q82) -> qapprox92
e -> qapprox114
```

TA4SP Output

At the end of this section , we give the output of TA4SP on this protocol. Notice that time computation is one hundred minutes (several optimisations could be done).

```
SUMMARY
   SAFE

DETAILS
   OVER_APPROXIMATION
   UNBOUNDED_NUMBER_OF_SESSIONS

PROTOCOL
   ./ViewOnlyProtocol.tif

GOAL
   SECRECY - As specified in HLPSL/IF

BACKEND
   TA4SP

COMMENTS
   For the given initial state, an over-approximation is used
   with an unbounded number of sessions.
   Terms supposed not to be known by the intruder are still secret.

STATISTICS
   Translation: 0.01 seconds
   Computation 6276.26 seconds


ATTACK TRACE
   No attack found
```