Automatic Compilation of Protocol Insecurity Problems into Logic Programming*

Alessandro Armando[1], Luca Compagna[1], and Yuliya Lierler[2]

[1] AI-Lab, DIST – Università di Genova,
{armando,compa}@dist.unige.it
[2] AI, Erlangen-Nürnberg Universität,
yuliya.lierler@informatik.uni-erlangen.de

Abstract. In this paper we show how protocol insecurity problems expressed in a multi-set rewriting formalism can be automatically translated into logic programming problems. The proposed translation paves the way to the construction of model-checkers for security protocols based on state-of-the-art solvers for logic programs. We have assessed the effectiveness of the approach by running the proposed reduction against a selection of insecurity problems drawn from the Clark & Jacob library of security protocols: by running state-of-the-art solvers against the resulting logic programming problems most of the (known) attacks on the considered protocols are found in a few seconds.

1 Introduction

Security protocols are communication protocols that aim at providing security guarantees (such as authentication of principals or secrecy of information) through the application of cryptographic primitives. In spite of their apparent simplicity security protocols are notoriously error-prone. Quite interestingly, many attacks can be carried out without breaking cryptography. These attacks exploit weaknesses in the protocol that are due to unexpected interleavings of different protocol sessions as well as to the possible interference of malicious agents. Since these weaknesses are very difficult to spot by simple inspection of the protocol specification, security protocols received growing attention by the Formal Methods and Automated Reasoning communities as a new, challenge application domain.

In the last decade we thus witnessed the development of a large number of new techniques for the analysis of security protocols. While some techniques (e.g., [21, 15, 7]) are tailored to the analysis of security protocols, others reduce

* We are grateful to Joohyung Lee and Vladimir Lifschitz for the comments and discussions related to the topic of the paper. This work was partially funded by FET Open EC Project "AVISPA: Automated Validation of Internet Security Protocols and Applications" (IST-2001-39252) and by the FIRB Project no. RBAU01P5SS. The last author was partially supported by Texas Higher Education Coordinating Board under Grant 003658-0322-2001.

the analysis of security protocols to some general purpose formalism such as CSP [14], rewriting logic [9], logic programming [1], or propositional logic [6]. While techniques in the first camp usually exhibit better performance, techniques in the second camp are normally simpler to adapt in response to changes or extensions to the underlying model.

In this paper we show how protocol insecurity problems expressed in a multi-set rewriting formalism can be automatically translated into logic programs with answer set semantics. The proposed translation paves the way to the construction of model-checkers for security protocols based on state-of-the-art solvers for logic programs. We implemented our ideas within SATMC,[3] a model-checker for security protocols developed in the context of the AVISPA Project,[4] a platform that aims at supporting the development of large-scale Internet security-sensitive protocols. We assessed the effectiveness of the approach by running the proposed reduction against a selection of problems drawn from Clark & Jacob library of security protocols [8]. By running a state-of-the-art answer set solver (e.g. CMODELS or [13] SMODELS [17]) against the resulting logic programming problems most of the (known) attacks on the considered protocols are found in a few seconds.

Outline of the paper. We start in Section 2 by introducing security protocols via a well-known (flawed) authentication protocol. In Section 3 and Section 4 we define the notions of protocol insecurity problems and logic programs, respectively. Section 5 is devoted to the description of the proposed reduction of protocol insecurity problems into logic programming. Experimental results are discussed in Section 6. We conclude in Section 7 with some final remarks.

2 The Needham-Schroeder Public-Key Protocol

Let us consider the well-known Needham-Schroeder Public-Key (NSPK) authentication protocol. In the common Alice&Bob notation, the NSPK protocol can be represented as:

$$\begin{array}{llcll} (1) & A & \rightarrow & B & : & \{A, Na\}_{Kb} \\ (2) & B & \rightarrow & A & : & \{Na, Nb\}_{Ka} \\ (3) & A & \rightarrow & B & : & \{Nb\}_{Kb} \end{array}$$

where A and B are the roles involved in the protocol; Ka and Kb are the public keys of A and B, respectively; Na and Nb are nonces[5] generated, respectively, by A and B. Step (1) of the protocol denotes A sending B a message comprising the identity of A and the nonce Na encrypted with Kb. Since $\{A, Na\}_{Kb}$ can only

[3] http://www.ai.dist.unige.it/satmc
[4] http://www.avispa-project.org
[5] *Nonces* are numbers randomly generated by principals and are intended to be used *only once*.

be deciphered by means of the private key Kb^{-1} and the latter is (by assumption) only known by B, then the effect of Step $(\mathit{1})$ is that only B can possibly learn the value of Na. In Step $(\mathit{2})$ agent B proves to A his participation in the protocol and, by sending Nb, asks A to do the same. In Step $(\mathit{3})$ agent A concludes by proving to B her own participation in the protocol. Thus successful execution of the NSPK protocol should give evidence to A and B that they talked to each other. The rationale is that, under the perfect cryptography assumption (see Section 3), only B and A could compose the appropriate response to the messages issued in $(\mathit{1})$ and $(\mathit{2})$, respectively.

Note that the above specification is parametric in the variables A, B, Ka, Kb, Na, Nb. Thus, if we denote the above protocol specification by $NSPK(A, B, Ka, Kb, Na, Nb)$, then $NSPK(alice, i, ka, ki, na, ni)$ and $NSPK(alice, bob, ka, kb, na2, nb)$ denote two sessions whereby $alice$ executes the protocol with i (the intruder) and bob respectively.[6] By using messages from one session to form messages in the other as illustrated below, i deceives bob into believing that he is talking to $alice$ whereas he is talking to i:

$$\begin{array}{llcll}
(\mathit{1.1}) & alice & \rightarrow & i & : \{alice, na\}_{ki} \\
(\mathit{2.1}) & i(alice) & \rightarrow & bob & : \{alice, na\}_{kb} \\
(\mathit{2.2}) & bob & \rightarrow & i(alice) & : \{na, nb\}_{ka} \\
(\mathit{1.2}) & i & \rightarrow & alice & : \{na, nb\}_{ka} \\
(\mathit{1.3}) & alice & \rightarrow & i & : \{nb\}_{ki} \\
(\mathit{2.3}) & i(alice) & \rightarrow & bob & : \{nb\}_{kb}
\end{array}$$

where $i(alice)$ indicates the intruder pretending to be $alice$.

3 Protocol Insecurity Problems

We model the concurrent execution of security protocols by means of a state transition system specified in declarative language based on multi-set rewriting [2]. In this paper we assume that the network is controlled by the very general Dolev-Yao intruder [10]. In this model the intruder has the ability to eavesdrop, divert, compose, decompose, encrypt, and decrypt messages. Furthermore we make the standard assumptions of *perfect cryptography* i.e. an encrypted message can be decrypted only with the appropriate decryption key, and of *strong typing* i.e. agents accept only well-typed messages.

A *protocol insecurity problem* is a tuple $\Xi = \langle \mathcal{F}, \mathcal{L}, \mathcal{R}, \mathcal{I}, \mathcal{G} \rangle$ where $\mathcal{F}$ is a set of atomic formulae of a sorted first-order language called *facts*, $\mathcal{L}$ is a set of function symbols called *rule labels*, and $\mathcal{R}$ is a set of (rewrite) rules of the form

[6] Here and in the rest of the paper we use capitalized identifiers for variables and lowercase identifiers for constants.

$L \xrightarrow{\ell} R$, where L and R are finite subsets of $\mathcal{F}$ such that the variables occurring in R occur also in L, and ℓ is an expression of the form $l(x)$, called *rule name*, where $l \in \mathcal{L}$ and x is the vector of variables obtained by ordering the variables occurring in L lexicographically. Let $L_1 \xrightarrow{\ell_1} R_1 \in \mathcal{R}$ and $L_2 \xrightarrow{\ell_2} R_2 \in \mathcal{R}$; we require that $\ell_1 = \ell_2$ if and only if $L_1 = L_2$ and $R_1 = R_2$. This additional requirement ensures that rule names are in one-to-one relation with the rewrite rules in $\mathcal{R}$; thus in the sequel we will often use rule names and rewrite rules interchangeably for the sake of brevity. $\mathcal{I}$ is a subset of $\mathcal{F}$ representing the initial state. In this setting, a *state* is denoted by the set of ground facts $S \subseteq \mathcal{F}$ that are true in it. (The ground facts that do not occur explicitly in S are considered to be false.) It is also possible to denote a state by the conjunction of facts that are true in it. Finally $\mathcal{G}$ is a boolean combination of facts in disjunctive normal form (DNF), whose disjuncts represent the bad states of the protocol.

Let S be a state and $(L \xrightarrow{\ell} R) \in \mathcal{R}$. If σ is a substitution such that $L\sigma \subseteq S$, then one possible next state is $S' = (S \setminus L\sigma) \cup R\sigma$ and we indicate this with $S \overset{\ell\sigma}{\rightsquigarrow} S'$. A *solution to (or attack on) a protocol insecurity problem* Ξ is a sequence of rules $\ell_0\sigma_1, \ldots, \ell_{k-1}\sigma_{k-1}$ where $\sigma_1, \ldots, \sigma_{k-1}$ are grounding substitutions[7] such that $S_i \overset{\ell_i\sigma_i}{\rightsquigarrow} S_{i+1}$ for $i = 0, \ldots, k-1$ with $S_0 = \mathcal{I}$ and $S_k \models \mathcal{G}$ (where $\models$ denotes entailment in classical logic).

It is convenient to relax the definition of the transition relation associated with a protocol insecurity problem by allowing parallel executions of rules while preserving the interleaving semantics. Let $L_1 \xrightarrow{\ell_1} R_1$ and $L_2 \xrightarrow{\ell_2} R_2$ be in $\mathcal{R}$ and let σ_1 and σ_2 be grounding substitutions such that $\ell_1\sigma_1 \neq \ell_2\sigma_2$. We say that $\ell_1\sigma_1$ *conflicts with* $\ell_2\sigma_2$ if and only if $L_1\sigma_1 \cap (L_2\sigma_2 \setminus R_2\sigma_2) \neq \emptyset$ or $L_2\sigma_2 \cap (L_1\sigma_1 \setminus R_1\sigma_1) \neq \emptyset$. Let S be a state, $L_i \xrightarrow{\ell_i} R_i \in \mathcal{R}$ for $i = 1, \ldots, m$, $L = (\bigcup_{i=1}^{m} L_i\sigma_i)$, $R = (\bigcup_{i=1}^{m} R_i\sigma_i)$, and $\Lambda = \{\ell_1\sigma_1, \ldots, \ell_m\sigma_m\}$, then we define $S \underset{\mathrm{P}}{\overset{\Lambda}{\rightsquigarrow}} S'$ if and only if $S' = ((S \setminus L) \cup R)$, $(L \setminus R) \cap (R \setminus L) = \emptyset$ and for each $i, j = 1, \ldots, m$ with $i \neq j$, we have that $\ell_i\sigma_i$ does not conflicts with $\ell_j\sigma_j$.

A *partial-order solution to (or partial-order attack on) a protocol insecurity problem* Ξ is a sequence of sets $\Lambda_0, \ldots, \Lambda_{k-1}$ such that $S_i \underset{\mathrm{P}}{\overset{\Lambda_i}{\rightsquigarrow}} S_{i+1}$ for $i = 0, \ldots, k-1$ with $S_0 = \mathcal{I}$ and $S_k \models \mathcal{G}$. The length of a partial-order attack is given by the number of sets in the sequence.

[7] A substitution is *grounding* if it maps every variable of the language to a ground term.

4 Logic Programming

A literal is an expression of the form a or $\neg a$ where a is an atom. A *logic rule* (lp-rule) is an expression

$$l_0 \leftarrow l_1, \ldots, l_m, not\ l_{m+1}, \ldots, not\ l_n \tag{1}$$

where l_0 is a literal or the symbol $\perp$ for falsehood, and $l_1, \ldots, l_n$ are literals for $0 \leq m \leq n$. The literal l_0 is called *head* of the lp-rule whereas $l_1, \ldots, l_m, not\ l_{m+1}, \ldots, not\ l_n$ is the *body*. If the head of an lp-rule is $\perp$ then the lp-rule is called *constraint* and it is written $\leftarrow l_1, \ldots, l_m, not\ l_{m+1}, \ldots, not\ l_n$. If the body of an lp-rule is empty then lp-rule is called *logic fact* (lp-fact). A *logic program* is a finite set of lp-rules.

We interpret logic programs via the answer set semantics [11, 12, 19]. Let Π be a logic program comprising lp-rules with $n = m$ (i.e. Π is a program without any occurrence of not) and let X be a consistent set of literals; we say that X is *closed* under Π if, for every lp-rule in Π, $l_0 \in X$ whenever $\{l_1, \ldots, l_m\} \subseteq X$. We say that X is an *answer set for* Π if X is the smallest set closed under Π. Now let Π be an arbitrary logic program and let X be a consistent set of literals. The *reduct* Π^X *of* Π *relative to* X is the set of rules $l_0 \leftarrow l_1, \ldots, l_m$ for all lp-rules in Π such that $X \cap \{l_{m+1}, \ldots, l_n\} = \emptyset$. Thus Π^X is a program without not. We say that X is an *answer set for* Π if X is an answer set for Π^X.

We finally extend the class of lp-rules with *choice lp-rules*, i.e. expressions of the form:

$$\{l_0\} \leftarrow l_1, \ldots, l_m, not\ l_{m+1}, \ldots, not\ l_n \tag{2}$$

where $l_0, \ldots, l_n$ are literals. For the lack of space here we do not provide the precise definition of the semantics of logic programs with choice lp-rules. (For a precise definition of an answer set for logic programs with choice lp-rules please see [18].) However, for the purposes of the present paper, it suffice to give the following, informal explanation. We say that the body of lp-rules of the form (1) or (2) is satisfied by X if $\{l_1, \ldots, l_m\} \subseteq X$ and $\{l_{m+1}, \ldots, l_n\} \cap X = \emptyset$. If, on the one hand, an lp-rule prescribes that if the body is satisfied by the answer set then its head must be in the answer set too. A choice lp-rule, on the other hand, prescribes that if its body is satisfied by the answer set, its head *may or may not* be in the answer set.

To illustrate, let us consider the program composed by lp-rules $a \leftarrow$, $\{b\} \leftarrow a$, and $c \leftarrow b$. It has two answer sets $\{a\}$ and $\{a, b, c\}$. Clearly the body of the first rule is satisfied therefore a shall be part of all answer sets. The body of the second choice lp-rule is also satisfied hence b may be in the answer set. The satisfaction of the body of the last rule depends on weather b is in the answer set or not. By adding the constraint $\leftarrow c$ to the example program we can eliminate the second answer set.

For simplicity in the sequel we will use the term lp-rules in a broad sense so to encompass also choice lp-rules.

5 Protocol Insecurity Problems as Logic Programs

Let $\Xi = \langle \mathcal{F}, \mathcal{L}, \mathcal{R}, \mathcal{I}, \mathcal{G} \rangle$ be a protocol insecurity problem with finite $\mathcal{F}$ and $\mathcal{R}$[8] and let k be a positive integer. In this section we will show how to build a logic program Π^k_Ξ such that any answer set of Π^k_Ξ corresponds to a partial-order attack on Ξ of length k. The basic idea of our translation is to add a time stamp i to rule names and facts. Facts are thus indexed by 0 through k and rule name by 0 through $k-1$. If p is a fact or a rule name and i is a time stamp in the appropriate range, then p^i is the corresponding atom. Program Π^k_Ξ is the union of the following lp-rules modeling the initial state, the goal, the execution of rewrite rules, the law of inertia, and mutual exclusion of conflicting rules.

For brevity we describe the translation by showing its application to the protocol insecurity problem Ξ_{NSPK} that models the two sessions of the NSPK authentication protocol informally introduced in Section 2. The facts in Ξ_{NSPK} are:

- $ik(T)$, meaning that the intruder knows the message T;
- $fresh(N)$, meaning that the nonce N has not been used yet;
- $m(J, S, R, T)$, meaning that S supposedly sent message T to R at step J;
- $w(J, S, R, [T_1, \ldots, T_k], C)$, meaning that R knows messages $T_1, \ldots, T_k$ at step J of session C, and—if $J \neq 0$—also that a message from S to R is awaited for step J of session C to be executed.

Initial State. The set $\mathcal{I}$ contains facts that encode the initial state. For each $f \in \mathcal{I}$ there is a corresponding lp-fact $f^0 \leftarrow$. For instance, the initial state of Ξ_{NSPK} is:[9]

$$w(0, a, a, [a, i, ka, ka^{-1}, ki], 1) \tag{3}$$

$$\mathbin{.} w(0, a, a, [a, b, ka, ka^{-1}, kb], 2) \mathbin{.} w(1, b, a, [b, a, kb, kb^{-1}, ka], 2) \tag{4}$$

$$\mathbin{.} fresh(nc(n1, 1)) \tag{5}$$

$$\mathbin{.} fresh(nc(n1, 2)) \mathbin{.} fresh(nc(n2, 2)) \tag{6}$$

$$\mathbin{.} ik(i) \mathbin{.} ik(a) \mathbin{.} ik(b) \mathbin{.} ik(ki) \mathbin{.} ik(ki^{-1}) \mathbin{.} ik(ka) \mathbin{.} ik(kb) \tag{7}$$

Fact (3) states that honest agent a plays the role of initiator in session 1 and knows her own identity, the agent she would like to talk with (the intruder), her public and private keys, and the intruder public key. Facts (4) represent the initial state of

[8] For simplicity of the description of translation we assume $\mathcal{F}$ and $\mathcal{R}$ to be ground.

[9] To improve readability we use the "." operator as set constructor. For instance, we write "$x \mathbin{.} y \mathbin{.} z$" to denote the set $\{x, y, z\}$.

the honest agents a and b and specify their involvement as initiator and responder (resp.) in session 2. Facts (5) and (6) state the initial freshness of the nonces. Facts (7) represent the information known by the intruder. The lp-rules corresponding to the initial state are:

$$w(0, a, a, [a, i, ka, ka^{-1}, ki], 1)^0 \leftarrow$$
$$\vdots$$
$$ik(kb)^0 \leftarrow$$

Goal. We recall that the goal if a formula in DNF specifying the set of bad states, whose reachability implies a violation of the desired security property. (For simplicity, here we assume that $\mathcal{G}$ contains only positive facts.) For each disjunct $g_1 \wedge \ldots \wedge g_n$ of $\mathcal{G}$ we generate the lp-rule $goal \leftarrow {g_1}^k, \ldots, {g_n}^k$. The constraint $\leftarrow$ *not goal* is also included in the output logic program. It restricts the answer sets to contain $goal$.

For example, successful execution of NSPK should ensure authentication of the responder with the initiator and vice versa. The attack situation can be easily modeled by the goal formula $w(1, a, b, [b, a, kb, kb^{-1}, ka], s(2)) \wedge w(0, a, a, [a, b, ka, ka^{-1}, kb], 2)$ that represents all the states in which b believes to have completed a session with a, while a did not start this session with him. The corresponding lp-rules are:

$$\begin{aligned} goal \leftarrow\ & w(1, a, b, [b, a, kb, kb^{-1}, ka], s(2))^i, \\ & w(0, a, a, [a, b, ka, ka^{-1}, kb], 2)^i \\ \leftarrow\ & not\ goal \end{aligned}$$

Inertia. For each $f \in \mathcal{F}$ there is an lp-rule of the form $f^{i+1} \leftarrow f^i, not\ \neg f^{i+1}$. Such lp-rule states that if some fact f is true at time i it is also true at time $i+1$ unless it is inferred to be false at time $i+1$. For instance, the inertia for the facts modeling the intruder knowledge is modeled by:

$$ik(T)^{i+1} \leftarrow ik(T)^i,\ not\ \neg ik(T)^{i+1}$$

Rewrite Rule Execution. For each rewrite rule $(L \xrightarrow{\ell} R) \in \mathcal{R}$ with $L = \{l_1, \ldots, l_m\}$, we generate the following lp-rules:

$\{\ell^i\} \leftarrow {l_1}^i, \ldots, {l_m}^i$,
$f^{i+1} \leftarrow \ell^i$, for each $f \in R \setminus L$, and
$\neg f^{i+1} \leftarrow \ell^i$, for each $f \in L \setminus R$.

The first rule states that if elements of L are satisfied at time i then rule ℓ might be applied at the same time. The last two rules state that if the rule ℓ is applied at time i then at $i+1$ the facts that belong to $R \setminus L$ hold and the ones which belong to $L \setminus R$ do not hold. In the last rule we introduce $\neg f^{i+1}$ for all facts $f \in L \setminus R$. When combined with the lp-rules modeling the inertia this forces f to not hold at time $i+1$.

We call the rules modeling the behavior of honest agents *protocol rules* and rules representing the intruder *intruder rules*. Here for the sake of brevity we present only one rewrite rule for each type and their corresponding compilation.

The following protocol rule models the activity of sending the first message of the NSPK:

$$
\begin{aligned}
fresh(nc(n1,S)) \mathbin{.} w(0,A,A,[A,B,Ka,Ka^{-1},Kb],S) &\xrightarrow{step_0(A,B,Ka,Kb,S)} \\
w(2,B,A,[nc(n1,S),A,B,Ka,Ka^{-1},Kb],S)& \\
\mathbin{.} m(1,A,B,\{A,nc(n1,S)\}_{Kb})&
\end{aligned}
$$

The lp-rules corresponding to this rewrite rule are:

$$
\begin{aligned}
\{step_0(A,B,Ka,Kb,S)^i\} &\leftarrow fresh(nc(n1,S))^i, \\
&\quad\ w(0,A,A,[A,B,Ka,Ka^{-1},Kb],S)^i \\
w(2,B,A,[nc(n1,S),A,B,Ka,Ka^{-1},Kb],S)^{i+1} &\leftarrow step_0(A,B,Ka,Kb,S)^i \\
m(1,A,B,\{A,nc(n1,S)\}_{Kb})^{i+1} &\leftarrow step_0(A,B,Ka,Kb,S)^i \\
\neg fresh(nc(n1,S))^{i+1} &\leftarrow step_0(A,B,Ka,Kb,S)^i \\
\neg w(0,A,A,[A,B,Ka,Ka^{-1},Kb],S)^{i+1} &\leftarrow step_0(A,B,Ka,Kb,S)^i
\end{aligned}
$$

The following intruder rule models the ability of the intruder to decrypt messages:

$$
ik(\{M\}_K) \mathbin{.} ik(K^{-1}) \xrightarrow{decrypt(K,M)} ik(\{M\}_K) \mathbin{.} ik(K^{-1}) \mathbin{.} ik(M) \tag{8}
$$

It states that if the intruder knows both the cypher-text $\{M\}_K$ and the decryption key K^{-1}, then he can learn M. The lp-rules corresponding to this rewrite rule are:

$$
\begin{aligned}
\{decrypt(K,M)^i\} &\leftarrow ik(\{M\}_K)^i, ik(K^{-1})^i \\
ik(M)^{i+1} &\leftarrow decrypt(K,M)^i
\end{aligned} \tag{9}
$$

Conflicts Exclusion. For each pair of conflicting rules ℓ_1 and ℓ_2 we generate a constraint of the form

$$
\leftarrow\ {\ell_1}^i, {\ell_2}^i
$$

In this way we forbid the parallel execution of conflicting rewrite rules.

For instance, $step_0(a, b, ka, kb, 2)$ and $step_0(a, i, ka, ki, 2)$ are conflicting rules in Ξ_{NSPK} since they would use simultaneously the same fresh nonce $nc(n1, 2)$. This is encoded in logic programming by means of the constraint $\leftarrow\ step_0(a, b, ka, kb, 2)^i, step_0(a, i, ka, ki, 2)^i$.

Let A is an answer set for the logic program Π^k_Ξ and let $\Lambda_i = \{\ell\sigma : \ell\sigma^i \in A \text{ and } (L \xrightarrow{\ell} R) \in \mathcal{R}\}$ for $i = 0, \ldots, k-1$, then it can be shown that $\Lambda_0, \ldots, \Lambda_{k-1}$ is a partial-order attack on Ξ.

It is worth pointing out that while the above reduction paves the way to an automatic compilation of protocol insecurity problems into logic programs, its direct application is not viable as the resulting logic programs can be of unmanageable size even for simple protocols. To overcome this difficulty in [3] we introduced a number of (attack preserving) optimizing transformations on protocols insecurity problems that make the approach both feasible and effective on many protocols of interest.

6 Experimental Results

SATMC is a SAT-based Model-Checker for security protocol analysis. Given a protocol insecurity problem Ξ, as a preliminary step SATMC applies the optimizing transformations presented in [3] to Ξ thereby obtaining a new protocol insecurity problem Ξ' such that any attack on Ξ' corresponds to an attack on Ξ and vice versa. Then SATMC generates a SAT formula $\Phi^k_{\Xi'}$ (for increasing values of k) such that any model of $\Phi^k_{\Xi'}$ corresponds to an attack on Ξ' (and hence to an attack on Ξ). Models of SAT formulae are automatically found by invoking state-of-the-art SAT solvers.

In order to assess the effectiveness of the reduction described in this paper we have developed a prototype implementation of the translation described in Section 5. As a consequence, SATMC can now also translate the optimized protocol insecurity problem Ξ' into a logic program $\Pi^k_{\Xi'}$ (for increasing values of k). The logic program $\Pi^k_{\Xi'}$ is then fed to the grounder LPARSE[10] and then to a state-of-the-art solver. (Currently both SMODELS and CMODELScan be used to this end.) Once an answer set is found, SATMC transforms it into a partial-order attack that is reported to the user.

We ran SATMC against a selection of flawed security protocols drawn from the Clark & Jacob library [8]. For each protocol we built a corresponding protocol insecurity problem modeling a scenario with a bounded number of sessions in which the involved principals exchange messages on a channel controlled by the most general intruder based on the Dolev-Yao model.

[10] `http://www.tcs.hut.fi/Software/smodels/lparse.ps.gz`

Table 1. Experimental results.

Pb	K	Reduction to Logic programming (**LP**) LP EncT (IF2LP / LPARSE)	LP ATs	LP Rules	CMODELS ATs	CMODELS CLs	CMODELS LP SolT	Reduction to SAT (**SAT**) EncT	ATs	CLs	SolT
ISO-CCF-1 U	4	0.10 / 0.28	< 1	2	< 1	1	0.02	0.04	< 1	< 1	0.00
ISO-CCF-2 M	4	0.18 / 1.12	2	7	2	3	0.05	0.32	< 1	6	0.01
ISO-PK-1 U	4	0.19 / 0.95	1	4	1	2	0.03	0.12	< 1	2	0.00
ISO-PK-2 M	4	0.50 / 3.87	4	17	4	4	0.14	0.89	2	17	0.00
ISO-SK-1 U	4	0.09 / 0.29	< 1	2	< 1	1	0.02	0.04	< 1	< 1	0.00
ISO-SK-2 M	4	0.25 / 1.65	4	11	4	4	0.11	0.38	< 1	3	0.00
NSPK	7	0.46 / 7.54	12	80	12	14	0.65	2.31	7	51	0.09
NSPK-server	8	2.74 / 26.54	17	198	17	19	1.47	8.03	9	212	0.22
SPLICE	9	1.79 / 72.44	27	333	27	32	2.47	4.63	14	91	0.21
Swick 1	5	0.97 / 14.81	11	36	11	11	0.32	1.03	4	17	0.02
Swick 2	6	1.63 / 51.81	16	148	16	18	1.11	3.18	8	59	0.08
Swick 3	4	0.38 / 24.36	6	12	6	7	0.13	0.82	5	12	0.02
Swick 4	5	1.85 / 137.98	20	62	20	21	0.59	11.05	15	64	0.18
Stubblebine rep	3	1.40 / 371.00	29	2,010	29	30	14.37	82.93	13	2,048	0.63

Table 1 reports the experimental results obtained by compiling to logic programs (**LP** sub-table) and by compiling to SAT[11] (**SAT** sub-table). We used CMODELS as solver for logic programs and Chaff [16] as SAT solver. Experiments were carried out on a PC with a 1.4 GHz CPU and 1 GB of RAM.

For each protocol we give the smallest value of k at which the attack is found (**K**). We also give the time spent for generating the logic program (IF2LP) and the time spent by LPARSE for grounding it (LPARSE),[12] the number of lp-atoms (**LP ATs**) and lp-rules (**LP Rules**) in the logic program (in thousands), the time spent by CMODELS for solving the logic program (**LP SolT**)[13] together with the number of atoms (**ATs**) and clauses (**CLs**) in the SAT formula generated by CMODELS in this solving phase (in thousands). Moreover, we give the time spent by SATMC for generating the SAT formula (**EncT**), the number of propositional variables (**ATs**) and clauses (**CLs**) in the SAT formula (in thousands), and the time spent by Chaff to solve the SAT formula (**SolT**).

[11] SATMC supports a variety of techniques for compiling protocol insecurity problems to SAT. The experiments described in this paper are obtained by using the linear encoding technique. See [4] for a survey of the encoding techniques supported by SATMC.

[12] The time spent for generating the ground logic program (**LP EncT**) is the sum of IF2LP and LPARSE.

[13] The results obtained by running SMODELS on this application domain are really comparable with those of CMODELS.

Since CMODELS[13] reduces the problem of finding answer sets of logic programs by reduction to SAT, it is interesting to compare the number of atoms and clauses directly generated by SATMC with that generated by CMODELSby compiling the logic program obtained by applying the reduction technique described in this paper. SATMC uses a noticeably smaller number of atoms, while the number of clauses generated often speaks in favor of CMODELS. The difference in the number of atoms can be explained by the advantage of the grounder of SATMC tuned to the protocol insecurity problems over the general purpose grounder LPARSE. As far as the number of clauses is concerned, the difference is due to the fact that CMODELS performs some reductions on the logic program before translating it into a propositional formula.[14]

As far are the timings are concerned, the experimental results indicate that the **SAT** approach outperforms the **LP** approach. But this is mainly due to the time spent by the grounder LPARSE that largely dominates the other times.

In [5] an optimized intruder model is proposed that leads in many cases to shorter attacks. The key idea is to model the ability of the intruder to decompose messages by means of axioms instead of rewrite rules. (An axiom is a formula that states a relation between the facts and that holds in all reachable states.) However, this requires non trivial extensions to the SAT-reduction techniques, whereas its application to the approach described in this paper is considerably simpler. For instance, the axiom

$$(ik(\{M\}_K) \wedge ik(K^{-1})) \supset ik(M) \tag{10}$$

states that, every time the intruder knows both a message encrypted with the key K and the decryption key K^{-1}, then he also knows M *at the very same time step*. This can me mimicked in logic programming by the lp-rules:

$$\begin{aligned} ik(M)^i &\leftarrow ik(\{M\}_K)^i, ik(K^{-1})^i \\ \neg ik(K^{-1})^i &\leftarrow \neg ik(M)^i, ik(\{M\}_K)^i \\ \neg ik(\{M\}_K)^i &\leftarrow \neg ik(M)^i, ik(K^{-1})^i \end{aligned} \tag{11}$$

By modeling the intruder rules by means of axioms and by changing the encoding described in this paper accordingly, preliminary experiments indicate that SATMC finds attacks which are up to 3 steps shorter therefby saving up to 44% atoms and clauses when applied to the problems in the Clark & Jacob library.

Axioms are also useful to model specific algebraic properties of cryptographic operators. For instance, the Diffie-Hellman protocol relies on the following property of exponentiation:

$$(g^X)^Y = (g^Y)^X$$

[14] The details on the reduction can be found at `http://www.cs.utexas.edu/users/tag/cmodels/cmodels-1.ps`

Such a property can be modeled as a set of axioms representing equivalence classes over facts. For instance, the axioms

$$ik((g^X)^Y) \supset ik((g^Y)^X) \qquad \text{and} \qquad ik((g^Y)^X) \supset ik((g^X)^Y)$$

state that $ik((g^X)^Y)$ and $ik((g^Y)^X)$ are in the same equivalence class. By adopting the above approach we have been able to generate with SATMC a logic program whose answer set corresponds to a (known) attack on the Diffie-Hellman protocol.

7 Conclusions

The work presented in [1] is closely related to ours. In that paper the authors put forward the idea of formalizing protocol insecurity problems (modeled according to Paulson's inductive model [20]) in logic programming and of using solvers for logic programs for automating the analysis of security protocols. In this paper we have described an approach to the automatic compilation of security protocol specifications (in a multi-set rewriting formalism) into logic programs. This reduction, combined with the optimizing transformations introduced in [3], paves the way to the construction of model-checkers for security protocols based on state-of-the-art solvers for logic programs. We have also thoroughly assessed the effectiveness of the proposed reduction by running our prototype implementation against a selection of flawed security protocols drawn from the Clark & Jacob library [8] and using CMODELS to solve the resulting logic programs. A comparison with the approach of compiling protocol insecurity problems into SAT, indicates that even if the reduction to SAT exhibits better performance, the reduction to logic programming can readily take into account specific algebraic properties of cryptographic operators. Moreover we expect a considerable gain in performance by using the SATMC grounder instead of LPARSE for grounding the resulting logic programs.

References

1. L. Carlucci Aiello and F. Massacci. Verifying security protocols as planning in logic programming. *ACM Trans. on Computational Logic*, 2(4):542–580, October 2001.
2. 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-Verlag, 2002.
3. A. Armando and L. Compagna. Automatic SAT-Compilation of Protocol Insecurity Problems via Reduction to Planning. In *Proc. of FORTE 2002*, LNCS 2529. Springer-Verlag, 2002.
4. A. Armando and L. Compagna. Abstraction-driven SAT-based Analysis of Security Protocols. In *Proc. of SAT 2003*, LNCS 2919. Springer-Verlag, 2003.
5. A. Armando and L. Compagna. An optimized intruder model for sat-based model-checking of security protocols. To appear in the Proc. of ARSPA 2004., 2004.

6. A. Armando, L. Compagna, and P. Ganty. SAT-based Model-Checking of Security Protocols using Planning Graph Analysis. In *Proc. of FME 2003*, LNCS 2805. Springer-Verlag, 2003.
7. D. Basin, S. Mödersheim, and L. Viganò. An On-The-Fly Model-Checker for Security Protocol Analysis. In *Proc. of ESORICS'03*, LNCS 2808. Springer-Verlag, 2003.
8. J. Clark and J. Jacob. A Survey of Authentication Protocol Literature: Version 1.0, 17. Nov. 1997. URL: `www.cs.york.ac.uk/~jac/papers/drareview.ps.gz`.
9. Grit Denker, Jose Meseguer, and Carolyn Talcott. Formal specification and analysis of active networks and communication protocols: The Maude experience. In *Proc. of DISCEX'00*, pages 251–265. IEEE Computer Society Press, 2000.
10. D. Dolev and A. Yao. On the Security of Public-Key Protocols. *IEEE Transactions on Information Theory*, 2(29), 1983.
11. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In Robert Kowalski and Kenneth Bowen, editors, *Logic Programming: Proc. Fifth Int'l Conf. and Symp.*, pages 1070–1080, 1988.
12. Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.
13. Yuliya Lierler and Marco Maratea. Cmodels-2: Sat-based answer set solver enhanced to non-tight programs. In *Proc. LPNMR-04*, pages 346–350, 2004.
14. G. Lowe. Casper: a Compiler for the Analysis of Security Protocols. *Journal of Computer Security*, 6(1):53–84, 1998.
15. J. K. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *Proc. of CCS'01*, pages 166–175, 2001.
16. M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*, 2001.
17. Ilkka Niemelä and Patrik Simons. Smodels—an implementation of the stable model and well-founded semanics for normal logic programs. In *Proc. 4th Int'l Conference on Logic Programming and Nonmonotonic Reasoning (Lecture Notes in Artificial Intelligence 1265)*, pages 420–429. Springer-Verlag, 1997.
18. Ilkka Niemelä and Patrik Simons. Extending the Smodels system with cardinality and weight constraints. In Jack Minker, editor, *Logic-Based Artificial Intelligence*, pages 491–521. Kluwer, 2000.
19. Ilkka Niemelä, Patrik Simons, and Timo Soininen. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138:181–234, 2002.
20. L. C. Paulson. The Inductive Approach to Verifying Cryptographic Protocols. *Journal of Computer Security*, 6(1):85–128, 1998.
21. D. Song. Athena: A new efficient automatic checker for security protocol analysis. In *Proc. of the 12th IEEE Computer Security Foundations Workshop (CSFW '99)*, pages 192–202. IEEE Computer Society Press, 1999.