# Abstraction-driven SAT-based Analysis of Security Protocols*

Alessandro Armando and Luca Compagna

DIST – Università degli Studi di Genova, Viale Causa 13 – 16145 Genova, Italy,
{armando,compa}@dist.unige.it

**Abstract.** In previous work we proposed an approach to the automatic translation of protocol insecurity problems into propositional logic with the ultimate goal of building an automatic model-checker for security protocols based on state-of-the-art SAT solvers. In this paper we present an improved procedure based on an abstraction/refinement strategy which, by interleaving the encoding and solving phases, leads to a significant improvement of the overall performance of our model-checker.

## 1 Introduction

In spite of their apparent simplicity, security protocols are notoriously error-prone. Many published protocols have been implemented and deployed in real applications only to be found flawed years later. For instance, the Needham-Schroeder authentication protocol [28] was found vulnerable to a serious attack 17 years after its publication [22]. The problem is that security protocols are intended to be used in open, hostile environments (such as, e.g., the Internet) and therefore steps carried out by honest participants interplay with the ability of malicious agents to eavesdrop, divert, and even forge new messages. This results in a combinatorially high number of actual protocol runs whose analysis stretches traditional validation techniques (e.g. testing). This problem has stimulated the development of formal method techniques for this novel application domain either by adapting existing techniques (such as, e.g., in [16, 29, 30]) or by devising new ones (such as, e.g., in [5, 7, 24, 12, 14, 20, 25]).

In [3] we proposed an approach to the automatic translation of protocol insecurity problems into propositional logic with the ultimate goal of building a model-checker for security protocols based on state-of-the-art SAT solvers. The approach combines a reduction of protocol insecurity problems to planning problems with well-known SAT-reduction techniques developed for planning. We have built SATMC, a model-checker for security protocols based on these ideas, which readily finds attacks on a set of well-known authentication protocols. Motivated by the observation that the time spent in generating the propositional formulae largely dominates the time spent by the SAT solver, in this paper

* We wish to thank Enrico Giuchiglia, Marco Maratea, Massimo Narizzano, and Armando Tacchella for many useful and stimulating discussions and their support in the use of the SIM solver.

we introduce an abstraction/refinement strategy that significantly improves the encoding time and hence the overall performance of the model-checker. We show that the performance of SATMC depends not only on the speed of the SAT solver but also on the ability of the SAT solver to return partial models.

*Structure of the paper.* In Section 2 we introduce the notion of protocol insecurity problem and show its relationship with the notion of planning problem. In Section 3 we introduce techniques for encoding bounded planning problems into SAT. In Section 4 we present our improved model-checking procedure based on abstraction and refinement. In Section 5 we present experimental results obtained with our model-checker which confirm the better performance of the improved procedure. In Section 6 we discuss the related work and finally, in Section 7, we draw some conclusions and outline the future work.

## 2 Protocol Insecurity Problems as Planning Problems

To illustrate, consider the following one-way authentication protocol:

$$\begin{array}{ll} (1) & A \rightarrow B : \{N\}_K \\ (2) & B \rightarrow A : \{f(N)\}_K \end{array}$$

where $N$ is a nonce[1] generated by Alice, $K$ is a symmetric key, $f$ is a function known to Alice and Bob, and $\{X\}_K$ denotes the result of encrypting text $X$ with key $K$. Successful execution of the protocol should convince Alice that she has been talking with Bob, since only Bob could have formed the appropriate response to the message issued in (*1*). In fact, Ivory can deceit Alice into believing that she is talking with Bob whereas she is talking with her. This is achieved by executing concurrently two sessions of the protocol and using messages from one session to form messages in the other as illustrated by the following protocol trace:

$$\begin{array}{ll} (1.1) & A \rightarrow I(B) : \{N\}_K \\ (2.1) & I(B) \rightarrow A : \{N\}_K \\ (2.2) & A \rightarrow I(B) : \{f(N)\}_K \\ (1.2) & I(B) \rightarrow A : \{f(N)\}_K \end{array}$$

Alice starts the protocol with message (*1.1*). Ivory intercepts the message and (pretending to be Bob) starts a second session with Alice by replaying the received message—cf. step (*2.1*). Alice replies to this message with message (*2.2*). But this is exactly the message Alice is waiting to receive in the first protocol session. This allows Ivory to finish the first session by using it—cf. (*1.2*). At the end of the above steps Alice believes she has been talking with Bob, but this is obviously not the case.

A problem with the rule-based notation used above to specify security protocols is that it leaves implicit many important details such as the shared information and how the principals should react to messages of an unexpected form.

[1] *Nonces* are numbers generated by principals that are intended to be used *only once*.

This kind of description is usually supplemented with explanations in natural language which in our case explain that $N$ is a nonce generated by Alice, that $f$ is a function known to the honest participants only, and that $K$ is a shared key. To cope with the above difficulties and pave the way to the formal analysis of protocols a set of models and specification formalisms have been put forward [8, 23]. In this paper we use a planning language *a la* STRIPS to this end.

A *planning problem* is a tuple $\Pi = \langle \mathcal{F}, \mathcal{A}, Ops, I, \mathcal{G} \rangle$, where $\mathcal{F}$ and $\mathcal{A}$ are disjoint sets of variable-free atomic formulae of a sorted first-order language called *fluents* and *actions* respectively; $Ops$ is a set of expressions of the form $(Pre \xrightarrow{Act} Add\,; Del)$ where $Act \in \mathcal{A}$ and $Pre$, $Add$, and $Del$ are finite sets of fluents such that $Add \cap Del = \emptyset$; $I$ is a set of fluents representing the initial state and $\mathcal{G}$ is a boolean combinations of fluents representing the final states. A state is represented by a set $S$ of fluents, meaning that all the fluents in $S$ hold in the state, while all the fluents in $\mathcal{F} \setminus S$ do not hold in the state (close-world-assumption). An action is applicable in a state $S$ iff the action preconditions (fluents in $Pre$) occur in $S$ and the application of the action leads to a new state obtained from $S$ by removing the fluents in $Del$ and adding those in $Add$. A *solution to a planning problem* $\Pi$, called *plan*, is a sequence of actions whose execution leads from the initial state to a final state and the preconditions of each action appears in the state to which it applies. Plans can be represented in a flexible way by means of partial-order plans. A *partial-order plan* is a pair $\langle \Lambda, \prec \rangle$ where $\Lambda \subseteq \mathcal{A} \times N$ and $\prec$ is a partial order on the set of natural numbers $N$. A plan $\mathcal{P}$ is in the set of the plans denoted by a partial-order plan $\langle \Lambda, \prec \rangle$ iff there exists a bijection $\pi : \Lambda \to \{1, \ldots, |\Lambda|\}$ such that if $i_1 \prec i_2$ then $\pi(\alpha_1, i_1) < \pi(\alpha_2, i_2)$; if $\pi(\alpha, i) = k$ then $\alpha$ is the $k$-th action in $\mathcal{P}$. The *length* of the partial-order plan $\langle \Lambda, \prec \rangle$ is the cardinality of the set $\{i \mid \exists \alpha \in \mathcal{A}\ s.t.\ \langle \alpha, i \rangle \in \Lambda\}$. For instance, the partial-order plan $\langle \{\langle a, 0\rangle, \langle b, 0\rangle, \langle c, 3\rangle, \langle d, 5\rangle, \langle a, 5\rangle\}, \{0 \prec 3, 0 \prec 5, 3 \prec 5\}\rangle$ has length 3 and represents the set of plans $\{\langle a, b, c, d, a\rangle, \langle b, a, c, d, a\rangle, \langle a, b, c, a, d\rangle, \langle b, a, c, a, d\rangle\}$.

A protocol insecurity problem is given by a transition relation specifying the run allowed by the protocol plus an initial state and a set of bad states (i.e. states whose reachability implies the violation of the desired security properties). Protocol insecurity problems can be easily recast as planning problems. The states of the transition system model the state of the honest principals, the knowledge of the intruder, as well as the messages sent over the channel but not yet processed by the intended recipient (or diverted by the intruder). Actions model the legal transitions that can be performed by honest participants as well as the abilities of the intruder.

For the simple protocol above, fluents are of the form:

- $i(t)$, meaning that the intruder knows the term $t$;
- $fresh(n)$, meaning that the term (usually is a constant representing a nonce) $n$ has not been used yet;
- $m(j, s, r, t)$, meaning that a message $t$ has been sent (supposedly) from principal $s$ to principal $r$ at protocol step $j$, and

– $w(j, s, r, [t_1, \ldots, t_k])$, representing the state of execution of principal $r$ at step $j$; it means that $r$ knows the terms $t_1, \ldots, t_k$ at step $j$, and (if $j = 1, 2$) that $r$ is waiting for a message from $s$ for step $j$ to be executed.

The initial state of the system is:[2]

$$w(0, a, a, [\,]) \,.\, w(1, a, b, [\,]) \,.\, w(0, b, b, [\,]) \,.\, w(1, b, a, [\,]) \\ .\, \mathit{fresh}(n_1) \,.\, \mathit{fresh}(n_2) \,.\, i(a) \,.\, i(b)$$

Fluents $w(0, a, a, [\,])$, $w(1, a, b, [\,])$, $w(0, b, b, [\,])$, and $w(1, b, a, [\,])$ state that principals $a$ and $b$ are ready to play both the role of the initiator and of the responder. Fluents $\mathit{fresh}(n_1)$ and $\mathit{fresh}(n_2)$ state that $n_1$ and $n_2$ are fresh numbers (i.e. they have never been used before) and then that they can be used as nonces. Finally $i(a)$ and $i(b)$ state that the identities of $a$ and $b$ are known to the intruder.

The behavior of the honest principals and of the intruder is specified by means of operators. The activity of sending the first message is modeled by:[3]

$$w(0, A, A, [\,]) \,.\, \mathit{fresh}(N) \xrightarrow{step_1(A,B,N)} m(1, A, B, \{N\}_k) \,.\, w(2, B, A, [f(N)]) \;; \\ w(0, A, A, [\,]) \,.\, \mathit{fresh}(N)$$

Intuitively, this operator says that, if in the current global state there is an initiator $A$ ready to start the protocol with somebody, and a fresh term $N$ is available, then $A$ can send the first message of the protocol to a responder $B$. By doing this, $A$ will update its state and, of course, $N$ will not be usable as fresh term anymore. Notice that nonce $f(N)$ is added to the acquired knowledge of $A$ for subsequent use. The receipt of the message and the reply of the responder is modeled by:

$$m(1, A, B, \{N\}_k) . w(1, A, B, [\,]) \xrightarrow{step_2(A,B,N)} m(2, B, A, \{f(N)\}_k) . w(3, B, B, [\,]) \;; \\ m(1, A, B, \{N\}_k) \,.\, w(1, A, B, [\,])$$

The final step of the protocol is modeled by:

$$m(2, B, A, \{f(N)\}_k) \,.\, w(2, B, A, [f(N)]) \xrightarrow{step_3(A,B,N)} w(4, A, A, [\,]) \;; \\ m(2, B, A, \{f(N)\}_k) \,.\, w(2, B, A, [f(N)])$$

where steps 3 and 4 occurring as first parameter in $w$-fluents are used to denote the final state of the responder and of the initiator, respectively.

The following rule models the ability of the intruder of diverting the information exchanged by the honest participants:

$$m(J, S, R, T) \xrightarrow{divert(J,R,S,T)} i(R) \,.\, i(S) \,.\, i(T) \;;\; m(J, S, R, T)$$

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

[3] Here and in sequel we use capital letters to denote variables.

The ability of encrypting and decrypting messages is modeled by:

$$i(T) \cdot i(K) \xrightarrow{encrypt(K,T)} i(\{T\}_K) \ ; \tag{1}$$

$$i(\{T\}_K) \cdot i(K) \xrightarrow{decrypt(K,T)} i(T) \ ; \tag{2}$$

Finally, the intruder can send arbitrary messages possibly faking somebody-else's identity in doing so:

$$i(T) \cdot i(S) \cdot i(R) \xrightarrow{fake(J,R,S,T)} m(J,S,R,T) \ ;$$

In particular, this operator states that, if the intruder knows a term $T$ and two agent identities $S$ and $R$, then it can send on the network channel the message $T$ to $R$ impersonating the identity of $S$. Notice that with the above rules we represent the most general intruder based on the Dolev-Yao model [15]. In this model the intruder has the ability of eavesdropping and diverting messages as well as that of composing, decomposing, encrypting, and decrypting messages (provided the encryption keys are known).[4] Finally, he can send those messages to other participants with a false identity.

A security protocol is intended to enjoy a specific security property. In our example this property is the ability of authenticating Bob to Alice. A security property can be specified by providing a set of "bad" states, i.e. states whose reachability implies a violation of the property. For instance, any state containing a subset of fluents of the form $w(4,A,A,[]) \cdot w(1,A,B,[])$ (i.e. A has finished a run of the protocol as initiator and B is still at the beginning of the protocol run as responder) witnesses a violation of the expected authentication property and therefore it should be considered as a bad state. It is easy to build a propositional formula $\mathcal{G}$ such that each model of $\mathcal{G}$ represents a bad state. For the above example $\mathcal{G} \equiv (w(4,a,a,[]) \wedge w(1,a,b,[])) \vee (w(4,b,b,[]) \wedge w(1,b,a,[]))$.

## 3 Automatic SAT-Compilation of Planning Problems

Let $\Pi = \langle \mathcal{F}, \mathcal{A}, Ops, I, \mathcal{G} \rangle$ be a planning problem with finite $\mathcal{F}$ and $\mathcal{A}$ and let $n$ be a positive integer, then it is possible to build a propositional formula $\Phi^n_\Pi$ such that any model of $\Phi^n_\Pi$ corresponds to a partial-order plan of length $n$ representing solutions of $\Pi$. The encoding of a planning problem into a set of SAT formulae can be done in a variety of ways (see [21, 17] for a survey). The basic idea is to add an additional time-index to actions and fluents to indicate the state at which the action begins or the fluent holds. Fluents are thus indexed by 0 through $n$ and actions by 0 through $n-1$. If $p$ is a fluent or an action and $i$ is an index in the appropriate range, then $i{:}p$ is the corresponding time-indexed propositional variable.

[4] In other words we do the *perfect encryption assumption*. It turns out that many protocols are flawed even under this strong assumption.

It is worth pointing out that the reduction of planning problems to SAT paves the way to an automatic SAT-compilation of protocol insecurity problems. However a direct application of the approach is not viable, because the resulting encodings are of unmanageable size even for simple protocols. To overcome this difficulty in [3] we put forward a number of optimizing transformations on protocols insecurity problems that make the approach both feasible and effective on many protocols of interest. For instance, certain rules can be merged together without loosing any possible attack on the protocol thereby decreasing the number of actions and hence the size of the resulting SAT encodings. A detailed description of the optimizing transformations goes beyond the scope of this paper and the interested reader should consult [3].

In the rest of this section we will formally describe the standard linear encoding and a combination of the previous one with the bitwise representation of actions.

### 3.1 Standard Linear Encoding

By using the standard linear encoding, $\Phi^n_\Pi$ is the smallest set (intended conjunctively) such that:

- **Initial State Axioms:** for all fluents $f$, if $p \in I$ then $0{:}p \in \Phi^n_\Pi$, else $\neg 0{:}p \in \Phi^n_\Pi$;
- **Goal State Axioms:** $\mathcal{G}_n \in \Phi^n_\Pi$, where $\mathcal{G}_n$ is the formula $\mathcal{G}$ in which each fluent occurring in it is indexed by $n$;
- **Universal Axioms:** for each $(Pre \xrightarrow{\alpha} Add\,; Del) \in Ops$ and $i = 0, \ldots, n-1$:

$$(i{:}\alpha \supset \bigwedge\{i{:}f \mid f \in Pre\}) \in \Phi^n_\Pi$$
$$(i{:}\alpha \supset \bigwedge\{(i+1){:}f \mid f \in Add\}) \in \Phi^n_\Pi$$
$$(i{:}\alpha \supset \bigwedge\{\neg(i+1){:}f \mid f \in Del\}) \in \Phi^n_\Pi$$

- **Explanatory Frame Axioms:** for all fluents $f$ and $i = 0, \ldots, n-1$:

$$(i{:}f \wedge \neg(i+1){:}f) \supset \bigvee\left\{i{:}\alpha \mid (Pre \xrightarrow{\alpha} Add\,; Del) \in Ops, f \in Del\right\} \in \Phi^n_\Pi$$
$$(\neg i{:}f \wedge (i+1){:}f) \supset \bigvee\left\{i{:}\alpha \mid (Pre \xrightarrow{\alpha} Add\,; Del) \in Ops, f \in Add\right\} \in \Phi^n_\Pi$$

- **Conflict Exclusion Axioms (CEA):** for $i = 0, \ldots, n-1$:

$$\neg(i{:}\alpha_1 \wedge i{:}\alpha_2) \in \Phi^n_\Pi$$

for all $\alpha_1 \neq \alpha_2$ such that $(Pre_1 \xrightarrow{\alpha_1} Add_1\,; Del_1) \in Ops$, $(Pre_2 \xrightarrow{\alpha_2} Add_2\,; Del_2) \in Ops$, and $Pre_1 \cap Del_2 \neq \emptyset$ or $Pre_2 \cap Del_1 \neq \emptyset$.

It is immediate to see that the number of literals in $\Phi^n_\Pi$ is in $O(n|\mathcal{F}|+n|\mathcal{A}|)$. Moreover the number of clauses generated by the Universal Axioms is in $O(nP_0|\mathcal{A}|)$ where $P_0$ is the maximal number of fluents mentioned in a list of an operator (usually a small number); the number of clauses generated by the Explanatory Frame Axioms is in $O(n|\mathcal{F}|)$; finally, the number of clauses generated by the CEA is in $O(n|\mathcal{A}|^2)$. For instance, the application of the above encoding to the *EKE* protocol (with $n = 5$) generates a propositional formula with 61,508 atoms and 13,948,534 clauses of whom 13,165,050 (about 94 %) are due to the CEA.

### 3.2 Linear Encoding combined with Bitwise Action Representation

Since Conflict Exclusion Axioms (CEA) grow quadratically in the number of actions, the application of the standard linear encoding to asynchronous concurrent systems can generate huge propositional formulae. The CEA are useful to restrict which actions may occur simultaneously and therefore to guarantee that any model of the propositional formula would correspond to a partial-order plan representing solutions (plans) of the original planning problem.

The bitwise action representation does not require CEA. In order to represent univocally each action in $\mathcal{A} = \{\alpha^1, \ldots, \alpha^{|\mathcal{A}|}\}$ $k = \lceil log_2|\mathcal{A}| \rceil$ propositional variables $(b^1, \ldots, b^k)$ are used. Let $\|\alpha^j\|$ the propositional formula whose model identifies the bitwise representation of the action $\alpha^j$, then the combination of the standard linear encoding with the Bitwise Action Representation (BAR) results in the propositional formula $\Phi_{\Pi}^n$ defined as in 3.1 where the conflict exclusion axioms are neglected and each occurrence of $i:\alpha$ is replaced by $i:\|\alpha\|$.[5] The number of literals in $\Phi_{\Pi}^n$ decreases, wrt the standard linear encoding, to $O(n|\mathcal{F}| + nk)$. However, while the number of clauses generated by the Universal Axioms does not change and the conflict exclusion axioms are not required, the number of clauses generated by the Explanatory Frame Axioms combined with the bitwise action representation becomes, in the worst case, exponential in the number of actions. Precisely, it is in $O(nk^{|\mathcal{A}|}|\mathcal{F}|)$.

To avoid this exponential growth it is sufficient to restore in the Explanatory Frame Axioms the propositional variables associated with the actions in place of their bitwise representation and to extend the formula as follow:

- **Bitwise Axioms:** for each $(Pre \xrightarrow{\alpha} Add\ ;\ Del) \in Ops$ and $i = 0, \ldots, n-1$:

$$i:\|\alpha\| \equiv i:\alpha \in \Phi_{\Pi}^n$$

With respect to the standard linear encoding, the number of literals in $\Phi_{\Pi}^n$ increases to $O(n|\mathcal{F}| + n|\mathcal{A}| + nk)$, and the number of clauses in $\Phi_{\Pi}^n$ is neither quadratic nor exponential in the number of actions. In fact the number of clauses generated by the Bitwise Axioms is in $O(n|\mathcal{A}| + nk|\mathcal{A}|)$. For instance, the application of this encoding to the *EKE* protocol with $n = 5$ generates a propositional formula with 61,578 atoms (among these 70 atoms are used to represent the actions) and 1,630,044 clauses (among these 846,560, i.e. about 52 %, are results from the bitwise axioms). From here on we call *bitwise encoding* this "smart" variant of the standard linear encoding combined with the bitwise action representation.

An important difference between the standard linear encoding and the bitwise encoding is that the former allows for some actions to be executed in parallel (at the same step), while the latter imposes that one (and only one) action can be executed at each step. Hence solutions found at step $\overline{n}$ by applying the standard linear encoding will be found at step $n \geq \overline{n}$ if the bitwise encoding is applied.

[5] Notice that $i:\|\alpha\|$ is the formula $\|\alpha\|$ in which each propositional variable occurring in it, is indexed by $i$.

## 4 The Abstraction/Refinement Strategy

Abstraction [11] is a powerful techniques for the analysis of large-state systems. The idea is to abstract the system under consideration into simpler one that that has all the behaviors of the original one, but may also exhibit some spurious behaviors. Thus—by construction—if a safety property holds for the abstract system, it will also hold for the concrete one. On the other hand, a counterexample for the abstract system does not always correspond to a counterexample for the concrete one. If the behavior that violates the safety property in the abstract system cannot be reproduced in the concrete system, the counterexample is said to be *spurious*. When such a counterexample is found, the abstraction must be refined in order to eliminate the spurious behavior. The procedure is then iterated until either a non spurious counterexample is found, or the abstract system satisfies the safety property.

More precisely, let $P$ be a set of propositional letters, we define a *labeled transition system* (LTS) to be a tuple $\langle \Sigma, \mathcal{I}, \mathcal{L}, \mathcal{R}, v \rangle$, where $\Sigma$ is a set of states, $\mathcal{I} \subseteq \Sigma$ is the set of initial states, $\mathcal{L}$ is a set of transition labels, $\mathcal{R} \subseteq \{s_0 \overset{l}{\rightsquigarrow} s_1 \mid s_0, s_1 \in \Sigma;\ l \in \mathcal{L}\}$ is the labeled transition relation[6], and $v$ is a total function mapping states into $2^P$. A *computation path of length* $k$ of a LTS is a sequence $s_0 \overset{l_1}{\rightsquigarrow} s_1 \cdots s_k \overset{l_{k+1}}{\rightsquigarrow} s_{k+1}$ such that $k \geq 0$, $s_0 \subseteq \mathcal{I}$, and $s_i \overset{l_{i+1}}{\rightsquigarrow} s_{i+1} \in \mathcal{R}$, for each $i = 0, \ldots, k$. Let $M = \langle \Sigma, \mathcal{I}, \mathcal{L}, \mathcal{R}, v \rangle$ and $M^a = \langle \Sigma^a, \mathcal{I}^a, \mathcal{L}^a, \mathcal{R}^a, v^a \rangle$ be two LTSs and let $h : \Sigma \to \Sigma^a$ be a total (abstraction) function, then $M^a$ *is an abstraction of* $M$ w.r.t. $h$, iff the following conditions hold:

- for every state $s$ in $\mathcal{I}$ there exists a state $s^a$ in $\mathcal{I}^a$ such that $h(s) = s^a$;
- for all states $s_0, s_1 \in \Sigma$ and $s_0^a \in \Sigma^a$ with $h(s_0) = s_0^a$, if $s_0 \overset{l}{\rightsquigarrow} s_1 \in \mathcal{R}$ then there exists a state $s_1^a \in \Sigma^a$ such that $s_0^a \overset{l^a}{\rightsquigarrow} s_1^a \in \mathcal{R}^a$ and $h(s_1) = s_1^a$.

If $M^a$ is an abstraction of $M$, then it is simple to prove that for every computation path $s_0 \overset{l_1}{\rightsquigarrow} \cdots \overset{l_{k+1}}{\rightsquigarrow} s_{k+1}$ of $M$ there exists a computation path $s_0^a \overset{l_1^a}{\rightsquigarrow} \cdots \overset{l_{k+1}^a}{\rightsquigarrow} s_{k+1}^a$ of $M^a$ such that $h(s_i) = s_i^a$, for each $i = 0, \ldots, k+1$. As a consequence, the following fact (called *preservation theorem*) holds: given a propositional formula $\varphi$ respecting the abstraction function $h$,[7] if $M^a \vDash AG\varphi$ then $M \vDash AG\varphi$.[8]

Since in the standard linear encoding, the number of CEA is the main source of difficulty it is tempting to avoid their generation altogether. It turns out that the resulting problem is an abstraction of the original problem. To show this, we must define the model-checking problem associated with a planning problem. Let $\Pi = \langle \mathcal{F}, \mathcal{A}, Ops, I, \mathcal{G} \rangle$ be a planning problem, then the model-checking problem associated with $\Pi$ is $M \vDash AG\neg\mathcal{G}$ where $AG\neg\mathcal{G}$ is the safety property to be checked and $M = \langle \Sigma, \mathcal{I}, \mathcal{L}, \mathcal{R}, v \rangle$ is the concrete LTS such that $v : \Sigma \to 2^{\mathcal{F}}$,[9]

[6] We restrict our attention to LTS with a total transition relation.

[7] Following [11] we say that *a propositional formula* $\varphi$ *respects an abstraction function* $h$ if for each $s \in \Sigma$, $v(h(s)) \vDash \varphi$ implies $v(s) \vDash \varphi$.

[8] Note that $A$ and $G$ represent the "for all computation" path quantifier and the "globally" temporal operator, respectively.

[9] Notice that, given a set $S$ with $2^S$ we indicate the powerset of $S$.

$\mathcal{I} = \{s \mid s \in \Sigma, v(s) = I\}$, $\mathcal{L} \subseteq 2^{\mathcal{A}}$ and $\mathcal{R}$ is the set of labeled transitions $s_0 \overset{\{\alpha_1,\ldots,\alpha_m\}}{\rightsquigarrow} s_1$ with $m \geq 0$ and such that the following conditions hold:[10]

**(1)** for each $i = 1, \ldots, m$, $(Pre_i \xrightarrow{\alpha_i} Add_i \; ; Del_i) \in Ops$;
**(2)** $\bigcup_{i=1}^{m} Pre_i \subseteq v(s_0)$;
**(3)** $v(s_1) = (v(s_0) \setminus \bigcup_{i=1}^{m} Del_i) \cup (\bigcup_{i=1}^{m} Add_i)$;
**(4)** for each $\alpha_i, \alpha_j \in \{\alpha_1, \ldots, \alpha_m\}$ such that $i \neq j$, then $Pre_i \cap Del_j = \emptyset$.

Intuitively, from a state $s_0$ it is possible to reach a state $s_1$ iff there exists a set of actions such that their preconditions hold in $s_0$ and for each pair of actions in that set, the conflict exclusion constraint is satisfied (i.e. the precondition list of the first action does not intersect with the delete list of the second and viceversa). By abstracting away the CEA we obtain the abstract LTS $M^a = \langle \Sigma, \mathcal{I}, \mathcal{L}^a, \mathcal{R}^a, v^a \rangle$ whose main difference from the concrete system $M$ lies in the labeled transition relation. In particular, since in the abstract system conflict exclusion is not enforced, $\mathcal{R}^a$ must satisfy only conditions (1), (2), and (3). Hence, all the computation paths allowed for the concrete system are also allowed in the abstract system (that is $\mathcal{R} \subseteq \mathcal{R}^a$) and the preservation theorem holds. On the other hand, if a counterexample $s_0 \overset{\Lambda_1}{\rightsquigarrow} \ldots \overset{\Lambda_{k+1}}{\rightsquigarrow} s_{k+1}$ ($s_0 \in \mathcal{I}$, $\Lambda_i$ is a set of actions for $i = 1, \ldots, k+1$, and $s_{k+1}$ is a bad state i.e. it violates $\neg\mathcal{G}$) is found in the abstract system, then, in order to not be spurious, it has to be validated in the concrete system. The validation procedure checks that each step of the counterexample does not involve conflicting actions i.e. it checks condition (4) over the sets $\Lambda_i$ ($i = 1, \ldots, k+1$). When the validation procedure fails, the abstract LTS must be refined by adding to its transition relation the conflict exclusion constraints thereby avoiding the conflicting actions discovered so far in spurious counterexamples. The whole procedure is repeated until either a counterexample without conflicting actions is found, or the abstract system satisfies the safety property.

## 5 A SAT-based Model-Checker for Security Protocols

We have implemented the above ideas in SATMC, a SAT-based model-checker for security protocol analysis. Given a protocol insecurity problem $\Xi$, SATMC starts by applying the aforementioned optimizing transformations to $\Xi$ thereby obtaining a new protocol insecurity problem $\Xi'$ which is then translated into a corresponding planning problem $\Pi_{\Xi'}$. $\Pi_{\Xi'}$ is in turn compiled into SAT using one of the methodologies outlined in Section 3 for increasing values of $n$ and the propositional formula generated at each step is fed to a state-of-the-art SAT solver (currently Chaff [27], SIM [18], and SATO [31] are currently supported). As soon as a satisfiable formula is found, the corresponding model is translated back into a partial order plan[11] which is reported to the user.

[10] If $m = 0$ then no action is applied and therefore $s_1 = s_0$, i.e. stuttering.
[11] The partial order plan represents attacks on the protocol.

We have run our tool against a selection of (flawed) security protocols drawn from the Clark/Jacob library [10]. For each protocol we have 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. Moreover, we assume perfect cryptography (see Section 2) and that all atoms are typed i.e. we exclude type confusion (*strong typing assumption*).[12] Note that since the number of sessions is bounded and strong typing is assumed, then an intruder with finite knowledge is sufficient to find all the possible attacks that can occur in this scenario.

The results of our experiments are reported in Table 1. Each protocol has been analyzed by applying one of the following encoding techniques:[13]

**CEA:** standard linear encoding with generation of the CEA enabled,
**BITWISE:** bitwise encoding, and
**NoCEA:** standard linear encoding with generation of the CEA disabled and therefore combined with the Abstraction Refinement Loop.

For each protocol we give the number of fluents (**F**) and of actions (**Acts**) in the correspondent planning problem, and for each encoding technique we give the smallest value of $n$ at which the attack is found (**N**), the number of propositional variables (**A**) and clauses (**CL**) in the SAT formula (in thousands), the time spent to generate the SAT formula (**EncT**), the time spent by SAT solver to solve the last SAT formula (**Last**), and the total time spent by the SAT solver to solve all the SAT formulae generated for that protocol (**Tot**).[14] In the context of **NoCEA** a comparison between Chaff and SIM has been performed and the solving times are shown together with the number of iterations of the Abstraction Refinement Loop (#).

When the **CEA** encoding technique is applied, encoding time largely dominates solving time and this is strictly related to the size of the SAT instances generated. Both the size and the time spent to generate the SAT formulae drop significantly if **BITWISE** is used. This enabled us to encode and find attacks on protocols (e.g. *Andrew*, *KaoChow 2*, *KaoChow 3*, and *Woo-Lam M*) that could not be solved by using the **CEA** encoding technique. However, finding an attack using **BITWISE** can require more iterative-deepening steps (cf. the **N** columns) than those required by applying the standard linear encoding. As a consequence on some instances, such as the *NSCK* and the *SPLICE* protocols, **CEA** performance better than **BITWISE** both in terms of the size the SAT formulae and in encoding time. Moreover, the addition of the bitwise axioms can significantly increase the solving time (see, e.g., *KaoChow 2* and *KaoChow 3* protocols).

[12] As pointed out in [19] type-flaw attacks can be prevented by tagging the fields of a message with information indicating its intended type. Therefore the strong typing assumption is allows us to restrict the search space and focus of the most interesting attacks.

[13] Experiments have been carried out on a PC with a 1.4 GHz CPU and 1 GB of RAM.

[14] Times are measured in seconds.

**Table 1.** Experiments on protocols from the Clark/Jacob library

| | | | CEA | | | | | BITWISE | | | | | NoCEA | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Protocol** | **F** | **Acts** | **N** | **A** (K) | **CL** (K) | **EncT** | **Chaff** **Last** | **Chaff** **Tot** | **N** | **A** (K) | **CL** (K) | **EncT** | **Chaff** **Last** | **Chaff** **Tot** | **N** | **A** (K) | **CL** (K) | **EncT** | **Chaff** **Last** | **Chaff** **Tot** | **Chaff** # | **SIM** **Last** | **SIM** **Tot** | **SIM** # |
| *Andrew* | 465 | 15,650 | 9 | 145 | - | - | - | - | 11 | 178 | 5,348 | 220.8 | 4.6 | 21.6 | 9 | 145 | 2,256 | 111.4 | 2.0 | 12.1 | 1 | 1.7 | 6.2 | 0 |
| *EKE* | 1,148 | 10,924 | 5 | 62 | 13,949 | 7,100 | 7.6 | 19.7 | 5 | 62 | 1,630 | 113.1 | 1.0 | 3.1 | 5 | 62 | 783 | 74.1 | 0.7 | 3.7 | 2 | 0.0 | 0.9 | 0 |
| *ISO-CCF-1 U* | 39 | 22 | 4 | $< 1$ | $< 1$ | 0.1 | 0.0 | 0.0 | 4 | $< 1$ | 1 | 0.1 | 0.0 | 0.0 | 4 | $< 1$ | $< 1$ | 0.1 | 0.0 | 0.0 | 0 | 0.0 | 0.0 | 0 |
| *ISO-CCF-2 M* | 81 | 132 | 4 | $< 1$ | 6 | 0.3 | 0.0 | 0.1 | 4 | $< 1$ | 10 | 0.5 | 0.0 | 0.0 | 4 | $< 1$ | 4 | 0.2 | 0.0 | 0.1 | 0 | 0.0 | 0.0 | 0 |
| *ISO-PK-1 U* | 61 | 49 | 4 | $< 1$ | 2 | 0.1 | 0.0 | 0.0 | 4 | $< 1$ | 3 | 0.2 | 0.0 | 0.0 | 4 | $< 1$ | 2 | 0.1 | 0.0 | 0.0 | 0 | 0.0 | 0.0 | 0 |
| *ISO-PK-2 M* | 125 | 279 | 4 | 2 | 17 | 0.9 | 0.0 | 0.0 | 4 | 2 | 22 | 1.1 | 0.0 | 0.0 | 4 | 2 | 10 | 0.6 | 0.1 | 0.1 | 0 | 0.0 | 0.1 | 0 |
| *ISO-SK-1 U* | 39 | 25 | 4 | $< 1$ | $< 1$ | 0.1 | 0.0 | 0.0 | 4 | $< 1$ | 1 | 0.1 | 0.0 | 0.0 | 4 | $< 1$ | $< 1$ | 0.1 | 0.0 | 0.0 | 1 | 0.1 | 0.1 | 0 |
| *ISO-SK-2 M* | 100 | 87 | 4 | $< 1$ | 3 | 0.4 | 0.0 | 0.0 | 4 | $< 1$ | 6 | 0.5 | 0.0 | 0.0 | 4 | $< 1$ | 3 | 0.4 | 0.0 | 0.0 | 0 | 0.0 | 0.0 | 0 |
| *KaoChow 1* | 2,753 | 2,042 | 7 | 36 | 355 | 18.4 | 0.1 | 0.7 | 8 | 41 | 345 | 15.8 | 0.9 | 1.6 | 7 | 36 | 131 | 8.0 | 0.1 | 0.7 | 4 | 0.2 | 0.6 | 0 |
| *KaoChow 2* | 32,963 | 22,344 | 9 | 531 | 35,178 | 1,494 | - | - | 11 | 642 | 6,244 | 309.7 | 180.9 | 473.1 | 9 | 531 | 1,804 | 140.4 | 1.6 | 15.6 | 5 | 9.2 | 26.4 | 0 |
| *KaoChow 3* | 49,378 | 55,727 | 9 | 995 | - | - | - | - | 11 | 1,206 | 17,528 | 1,009.9 | 303.4 | 1,063.5 | 9 | 995 | 5,737 | 585.5 | 7.5 | 41.6 | 1 | 28.5 | 127.6 | 3 |
| *NSCK* | 6,755 | 5,220 | 9 | 115 | 787 | 41.3 | 0.4 | 1.6 | 10 | 127 | 1,131 | 46.3 | 1.3 | 4.6 | 9 | 115 | 334 | 17.1 | 0.3 | 1.3 | 0 | 0.4 | 1.5 | 0 |
| *NSPK* | 429 | 662 | 7 | 7 | 51 | 2.3 | 0.1 | 0.1 | 7 | 7 | 83 | 3.4 | 0.1 | 0.2 | 7 | 7 | 33 | 1.5 | 0.1 | 0.1 | 0 | 0.1 | 0.1 | 0 |
| *NSPK-server* | 298 | 604 | 8 | 9 | 212 | 8.0 | 0.1 | 0.2 | 8 | 9 | 115 | 5.1 | 0.1 | 0.2 | 8 | 9 | 54 | 2.8 | 0.1 | 0.1 | 0 | 0.1 | 0.2 | 0 |
| *SPLICE* | 822 | 658 | 9 | 14 | 91 | 4.6 | 0.1 | 0.2 | 11 | 17 | 160 | 6.9 | 0.2 | 1.5 | 9 | 14 | 62 | 3.6 | 0.1 | 0.2 | 0 | 0.1 | 0.3 | 1 |
| *Swick 1* | 496 | 258 | 5 | 4 | 17 | 1.0 | 0.1 | 0.1 | 7 | 6 | 38 | 1.7 | 0.0 | 0.1 | 5 | 4 | 13 | 0.8 | 0.0 | 0.1 | 1 | 0.1 | 0.1 | 0 |
| *Swick 2* | 693 | 450 | 6 | 8 | 59 | 3.2 | 0.1 | 0.1 | 7 | 9 | 65 | 2.9 | 0.0 | 0.1 | 6 | 8 | 29 | 1.7 | 0.1 | 0.1 | 0 | 0.1 | 0.1 | 0 |
| *Swick 3* | 526 | 482 | 4 | 5 | 12 | 0.8 | 0.1 | 0.1 | 6 | 7 | 46 | 2.0 | 0.0 | 0.1 | 4 | 5 | 11 | 0.7 | 0.0 | 0.1 | 1 | 0.1 | 0.1 | 0 |
| *Swick 4* | 1,357 | 1,283 | 5 | 15 | 64 | 11.0 | 0.1 | 0.2 | 6 | 17 | 162 | 15.9 | 0.2 | 0.7 | 5 | 15 | 57 | 10.2 | 0.1 | 0.3 | 1 | 0.3 | 0.5 | 0 |
| *Stubblebine rep* | 1,409 | 2,408 | 3 | 13 | 2,048 | 82.9 | 0.3 | 0.6 | 3 | 13 | 194 | 10.8 | 0.1 | 0.2 | 3 | 13 | 95 | 6.3 | 0.1 | 0.1 | 1 | 0.1 | 0.1 | 0 |
| *Woo-Lam M* | 45,584 | 27,051 | 6 | 481 | - | - | - | - | 7 | 554 | 5,977 | 433.6 | 4.0 | 12.4 | 6 | 481 | 2,498 | 304.4 | 1.8 | 7.7 | 1 | 2.3 | 5.6 | 0 |

(K) indicates that the value in this column are given in thousands;
- means that a memory out has been reached; and
$< 1$ means that the number of atoms or clauses is less than 1 thousand;

To cope with the above issues it is convenient to apply the **NoCEA** encoding technique that is based on the *Abstraction Refinement Loop* described in Section 4. Of course, by disabling the generation of the conflict exclusion axioms, we are no longer guaranteed that the solutions found are linearizable and hence executable.[15] SATMC therefore looks for conflicting actions in the partial order plan found and extends the previously generated encoding with clauses negating the conflicts (if any). The resulting formula is then fed back to the SAT-solver and the whole procedure is iterated until a solution without conflicts is met or the formula becomes unsatisfiable.

The **NoCEA** encoding technique performs uniformly better on all the analyzed protocols. Another interesting point is that while the time spent by Chaff and SIM to solve the propositional formulae is comparable in most cases (see the columns **Last** in the table **NoCEA**),[16] the number of iterations of the Abstraction Refinement Loop is in most of the cases smaller when SIM is used. This is due to the different strategies implemented by the two solvers. Both solvers incrementally build a satisfying assignment, but while SIM terminates (possibly returning a partial assignment) as soon as all the clauses are subsumed by the current satisfying assignment, Chaff avoids the subsumption check for efficiency reasons and thus halts only when the assignment is total. As a consequence the partial order plans found using Chaff usually contain more actions and thus are likely to have more conflicts. This explains why SIM is in many cases more effective than Chaff.

SATMC is one of the back-ends of the AVISS tool [2]. Using the tool, the user can specify a protocol and the security properties to be checked using a high-level specification language similar to the Alice&Bob notation we used in Section 1 to present our simple authentication protocol. The AVISS tool translates the specification into a rewrite-based declarative Intermediate Format (IF) based on multiset rewriting which is amenable to formal analysis. SATMC can optionally accept protocol specifications in the IF language which are then automatically translated into equivalent planning problems. Thus, by using SATMC in combination with the AVISS tool, the user is relieved from the time-consuming and error-prone activity of providing a detailed specification of the protocol insecurity problem in terms of a planning problem.

## 6 Related Work

The idea of regarding security protocols as planning problems is not new. An executable planning specification language $\mathcal{AL}_{SP}$ for representing security protocols and checking the possibility of attacks via a model finder for logic programs with stable model semantics has been proposed in [1]. Compared to this approach SATMC performs better (on the available results) and can readily exploit improvements of state-of-the-art SAT solvers.

[15] A counterexample in the model checking problem corresponds to a model of the propositional formula generated and, therefore, to solutions of the planning problem.

[16] Notice that Chaff performs better on big instances.

Gavin Lowe and his group at the University of Leicester (UK) have analyzed problems from the Clark/Jacob library [10] using Casper/FDR2 [16]. This approach has been very successful for discovering new flaws in protocols. However, first experiments on the search time indicate that SATMC is more effective than Casper/FDR2. Besides that Casper/FDR2 limits the size of messages that are sent in the network and is not able to handle non-atomic keys.

The Murphi model-checker has been used in [26] for analyzing some cryptographic protocols such as the Needham-Schroeder Public-Key. Experimental results indicate that Murphi suffers from state-space explosion. To cope with this problem several restrictions on the model are put in place. For instance, the size of the channel is fixed a priori.

In the context of the AVISS tool we have performed a thorough comparison between the back-ends integrated in it. The On-the-Fly Model-Checker (OFMC) [6] performs uniformly well on all the Clark/Jacob library. However, it is interesting to observe that in many cases the time spent by the SAT solver is equal to or even smaller than the time spent by OFMC for the same protocol. The Constraint-Logic-based model-checker (CL) [9] is able to find type-flaw attacks (as well as OFMC). However, the overall timing of SATMC is better than that of CL. Detailed results about these experiments can be found in [2].

## 7 Conclusions and Perspectives

The work presented in this paper is part of a larger effort aiming at the construction of an industrial-strength SAT-based model-checker for security protocols. In this paper we have introduced an abstraction/refinement strategy which, by interleaving encoding and solving, allows to halve the size of the propositional encoding for the most complex protocols we have analyzed so far. Since even bigger savings are expected on more complex protocols, the improved procedure presented in this paper paves the way to the analysis of complex protocols (e.g. e-commerce protocols) arising in real-world applications.

As far as the role of the SAT-solver is concerned, our experiments using Chaff and SIM indicate that the ability to return partial models can be as important as the pure solving time. In the future work we plan to tighten the integration with the SAT-solver in such a way that clauses generated by the refinement phase are "learnt" by the solver, thereby directly exploiting the sophisticated search strategies incorporated into the solver. However, since the solving time is still dominated by the encoding time, our current focus is on finding ways to reduce the latter.

Experimental results led with COMPACT [13] indicate that the application of propositional simplification techniques reduces dramatically the dimension of the encodings thereby suggesting that they suffer from some form of redundancy. A close look to the problem reveals that this is due to the fact that linear encodings do not exploit the knowledge about the initial state and therefore they encode protocol behaviors which can be safely neglected. This led us to consider a more sophisticated encoding technique, graphplan-based encoding [21], which leads

to even smaller propositional encodings [4]. Notice however that the greater generality of linear encodings is useful when analyzing protocols w.r.t. partially specified initial states.

Another promising approach amounts to treating properties of cryptographic operations as invariants. Currently these properties are modeled as actions (cf. rules (1) and (2) in Section 2) and this has a bad impact on the size of the final encoding. A more natural way to deal with these properties amounts to building them into the encoding but this requires, among other things, a modification of the explanatory frame axioms.

## 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, 2001.
2. A. Armando, D. Basin, M. Bouallagui, Y. Chevalier, L. Compagna, S. Moedersheim, M. Rusinowitch, M. Turuani, L. Viganò, and L. Vigneron. The AVISS Security Protocols Analysis Tool. In *Proc. CAV'02*. 2002.
3. A. Armando and L. Compagna. Automatic SAT-Compilation of Protocol Insecurity Problems via Reduction to Planning. In *Intl. Conf. on Formal Techniques for Networked and Distributed Systems (FORTE)*, Houston, Texas, November 2002.
4. A. Armando, Compagna. L., and P. Ganty. Sat-based model-checking of security protocols. In *Proc.of Formal Methods Europe 2003 (FME03)*, Pisa, Sept. 2003.
5. David Basin and Grit Denker. Maude versus haskell: an experimental comparison in security protocol analysis. In Kokichi Futatsugi, editor, *Electronic Notes in Theoretical Computer Science*, volume 36. Elsevier Science Publishers, 2001.
6. David Basin, Sebastian Moedersheim, and Luca Viganò. An On-the-Fly Model-Checker for security protocol analysis. Forthcoming, 2003.
7. D. Bolignano. Towards the formal verification of electronic commerce protocols. In *Proc.of the IEEE Computer Security Foundations Workshop*, pages 133–146. 1997.
8. Cervesato, Durgin, Mitchell, Lincoln, and Scedrov. Relating strands and multiset rewriting for security protocol analysis. In *PCSFW: Proceedings of The 13th Computer Security Foundations Workshop*. IEEE Computer Society Press, 2000.
9. Y. Chevalier and L. Vigneron. A Tool for Lazy Verification of Security Protocols. In *Proceedings of the Automated Software Engineering Conference (ASE'01)*. IEEE Computer Society Press, 2001.
10. John Clark and Jeremy Jacob. A Survey of Authentication Protocol Literature: Version 1.0, 17. Nov. 1997. URL `http://www.cs.york.ac.uk/~jac/papers/drareview.ps.gz`.
11. E. Clarke, A. Gupta, J. Kukula, and O. Strichman. SAT based abstraction-refinement using ILP and machine learning techniques. In *Proc. of Conference on Computer-Aided Verification (CAV'02)*, LNCS, 2002.
12. Ernie Cohen. TAPS: A first-order verifier for cryptographic protocols. In *Proc. of The 13th Computer Security Foundations Workshop*, 2000.
13. James M. Crawford and L. D. Anton. Experimental results on the crossover point in satisfiability problems. In Richard Fikes and Wendy Lehnert, editors, *Proceedings of the 11th AAAI Conference*, pages 21–27, California, 1993. AAAI Press.
14. Grit Denker, Jonathan Millen, and Harald Rueß. The CAPSL Integrated Protocol Environment. Technical Report SRI-CSL-2000-02, SRI International, Menlo Park, CA, October 2000. Available at `http://www.csl.sri.com/~millen/capsl/`.

15. Danny Dolev and Andrew Yao. On the security of public-key protocols. *IEEE Transactions on Information Theory*, 2(29), 1983.
16. B. Donovan, P. Norris, and G. Lowe. Analyzing a library of security protocols using Casper and FDR. In *Proceedings of the Workshop on Formal Methods and Security Protocols*. 1999.
17. Michael D. Ernst, Todd D. Millstein, and Daniel S. Weld. Automatic SAT-compilation of planning problems. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 1169–1177. Morgan Kaufmann Publishers, San Francisco, 1997.
18. E. Giunchiglia, M. Maratea, A. Tacchella, and D. Zambonin. Evaluating search heuristics and optimization techniques in propositional satisfiability. In *Proceedings of IJCAR'2001*, pages 347–363. Springer-Verlag, 2001.
19. Heather, Lowe, and Schneider. How to prevent type flaw attacks on security protocols. In *PCSFW: Proceedings of The 13th Computer Security Foundations Workshop*. IEEE Computer Society Press, 2000.
20. Florent Jacquemard, Michael Rusinowitch, and Laurent Vigneron. Compiling and Verifying Security Protocols. In M. Parigot and A. Voronkov, editors, *Proceedings of LPAR 2000*, LNCS 1955, pages 131–160. Springer-Verlag, Heidelberg, 2000.
21. Henry Kautz, David McAllester, and Bart Selman. Encoding plans in propositional logic. In *KR'96: Principles of Knowledge Representation and Reasoning*, pages 374–384. Morgan Kaufmann, San Francisco, California, 1996.
22. G. Lowe. Breaking and fixing the Needham-Shroeder public-key protocol using FDR. In *Proceedings of TACAS'96*, pages 147–166. Springer-Verlag, 1996.
23. Gawin Lowe. Casper: a compiler for the analysis of security protocols. *Journal of Computer Security*, 6(1):53–84, 1998.
24. W. Marrero, E. Clarke, and S. Jha. Model checking for security protocols, 1997.
25. Catherine Meadows. The NRL protocol analyzer: An overview. *Journal of Logic Programming*, 26(2):113–131, 1996.
26. J.C. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using murphi. In *Proc.of IEEE Symposium on Security and Privacy*, pages 141–153. 1997.
27. Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*. 2001.
28. R. M. (Roger Michael) Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. Technical Report CSL-78-4, Xerox Palo Alto Research Center, Palo Alto, CA, USA, 1978. Reprinted June 1982.
29. L.C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1):85–128, 1998.
30. D. Song. Athena: A new efficient automatic checker for security protocol analysis. In *Proceedings of the 12th IEEE Computer Security Foundations Workshop (CSFW '99)*, pages 192–202. IEEE Computer Society Press, 1999.
31. H. Zhang. SATO: An efficient propositional prover. In William McCune, editor, *Proceedings of CADE 14*, LNAI 1249, pages 272–275. Springer-Verlag, 1997.