Efficient Infinite-State Analysis of Security Protocols

Antti Huima (huima@ssh.fi)
SSH Communications Security, Ltd.
Tekniikantie 12, FIN-02150 ESPOO, FINLAND

Abstract

We propose a new method and present a tool for the analysis of cryptographic protocols. The method is based on symbolic state space search. It can be used to analyze thoroughly an infinite state space if the infiniteness is caused only by the infiniteness of the enemy but not by an unbounded number of interleaved protocol runs nor unbounded behaviours of single protocol participants. The method is complete for the class of protocols it is defined for and does not require user interaction to work.

1 Introduction

In this paper we consider the problem of analyzing cryptographic protocols by using symbolic state space enumeration and model checking. State space enumeration is the act of generating explicitly the state graph of a given system. By model checking we mean the act of verifying that the generated state graph has a certain structure defined by logical formulae. Symbolic state space enumeration is an extension of explicit state space enumeration: individual states are not enumerated but state *classes*. Thus, a symbolic state graph is a graph whose vertices denote sets of states. We warn the reader that the phrase 'symbolic model checking' has various, slightly different interpretations. For example, one could associate symbolic model checking automatically with binary decision diagrams (BDDs) [2]; we do not, however, use BDDs.

1.1 Related Work

Model checking security protocols has received a lot of attention recently. Lowe and Roscoe have used the CSP [13] model checker FDR [9, 31] to perform finite-state analysis of cryptographic protocols [18, 21, 23]. Lowe has also considered the problem of obtaining results for an unbounded number of participants [20] and presented a compiler for translating high-level protocol descriptions into CSP formulae [19]. Steve Schneider has also examined the use of CSP [32].

Mitchell et al have applied the generic-purpose verification tool Murφ to the finite-state analysis of cryptographic protocols [27, 28, 33]. Other generic tools have been used also, such as Astral [6] and the LOTOS tools [1, 10].

Clarke, Jha and Marrero have developed a special-purpose model checker for cryptographic protocols [5]. It also searches a finite state space.

There exists also the famous NRL protocol analyzer [23, 24]. It performs backwards reachability analysis for a system consisting of an unbounded number of participants. It is thus examining an infinite state space but its algorithm does not always terminate. Millen's Interrogator [15, 25] is another tool for doing backwards analysis.

General information about finite-state analysis of cryptographic protocols can be found e.g. from [26].

1.2 Other Methods

There exist also other methods for the formal analysis of cryptographic protocols, for example the BAN logic [3, 4] and its derivatives [11, 36, 37], and Paulson's inductive method [30]. Many survey articles can be found in the literature [12, 15, 22].

1.3 Overview

In this paper we present HPA1 (Huima's Protocol Analyzer 1), our new method and tool for analyzing cryptographic protocols.

Our underlying model resembles very much that of [5]. However, the method presented therein and most of the other methods confine to a finite state space. In contrast to this, we are able to analyze the whole infinite state space generated by a limited number of participants. We accomplish this by using symbolic state space enumeration.

In a sense, our method lifts that of [5] to a symbolic level and extends it. However, we developed our method independently. The similarities to [5] are purely coincidental.

The rest of this paper is structured as follows.

- In the next section, we examine different types of infinite state spaces that occur in the formal analysis of cryptographic protocols, relating the ideas to our method.

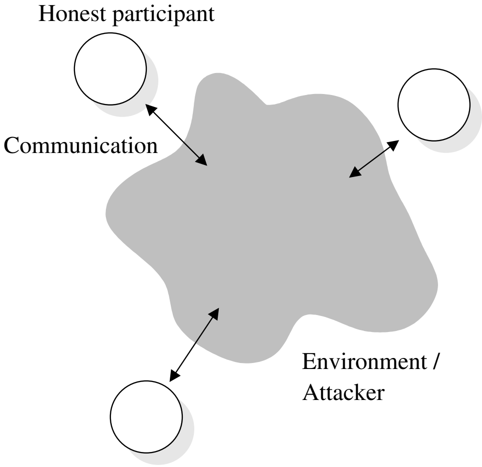

Figure 1: The worst-case model.

- In Sec. 3, we give an overview of our method.
- The formal model, algorithms and optimizations of the analysis process are discussed in Sec. 4.
- We discuss our implementation of the method and present a canonical example in Sec. 5.
- In the last two sections we give directions for future research and draw conclusions.

2 Infinite State Spaces And Protocol Models

An often used and theoretically appealing model for cryptographic protocols is the following: there is a set of participants P and an environment E. All the participants have a local state. The participants are able to (1) modify their local states indenpendently of the rest of the system, (2) send messages to the environment and (3) receive messages from the environment. This is all the participants can do. For example, two participants cannot communicate directly, bypassing the environment. Thus, all messages go through the environment *that models the attacker at the same time*. This is the 'worst-case model' whose origins can be traced to Dolev and Yao [8]. See Fig. 1.

In this model the global state of the system can be given as the sum of the local states of the participants P and the messages the environment E has seen. The environment does not need to have an explicit control state of its own because no assumptions concerning the behaviour of the environment are made. We can identify the participants with their local states and the environment with its state, i.e. the set of messages known to it. Thus the global state can be given as $\langle P,E\rangle$, where P is a set of local states and E is a set of messages.

We can define a transition relation $\rightarrow$ between global states such that $\langle P,E\rangle \rightarrow \langle P',E'\rangle$ iff the latter state is obtained from the former when a single participant executes exactly one step in $\langle P,E\rangle$, either (1) changing its own local state, (2) sending a message to the environment or (3) receiving a message from the environment.

Let us now consider the state space generated by an initial state $\langle P_0,E_0\rangle$ and the transition relation $\rightarrow$, that is, the smallest set of global states that contains the initial state and is closed under $\rightarrow$. The state space can be infinite if at least one of the following holds:

A. Some participants are able to perform an unbounded number of steps.

B. Some participants have the option of entering into infinitely many distinct control states during a local step.

C. The set of participants P is infinite.

D. The set of messages a participant can receive at some step is infinite.

We make here a distinction between the control state of a participant and the set of messages it has received, processed and sent. This distinction can be stated as follows: two local states contain the same control state iff the states become identical when all cryptographic messages are considered equivalent.

In general, each one of the factors A–D leads to an infinite state space, although pathological counter-examples are easy to conceive.

In the literature it is usually assumed that A and B do not hold. In other words, the behaviours of single participants are restricted to finite programs. Most models assume that C does not hold, the NRL protocol analyzer [24] and Paulson's method [30] being some of the prominent exceptions. The following proposition gives perspective to the condition C.

Proposition 2.1 *Protocol analysis is undecidable for system consisting of an unbounded number of participants.*

(Proof sketch) For the usually employed protocol models it is straightfoward to prove that an infinite collection of participants is collaboratively able to simulate an arbitrary Turing machine [16] arbitrarily long. Thus Turing-completeness can be achieved. □

Motivated by these observations we introduce the following terminology.

Definition 2.1 (Model Types) Let M be a protocol model that is capable of generating infinite state spaces. If the conditions A–C do not hold for M, and the infiniteness of the state spaces is caused by the condition D only, the model is of type I. Otherwise it is of type II. If a model generates finite state spaces only it is of type 0.

The NRL protocol analyzer contains a type II model. As mentioned previously, its algorithm does not always terminate. This is an unavoidable consequence of Prop. 2.1. Most other state space search based approaches employ a model

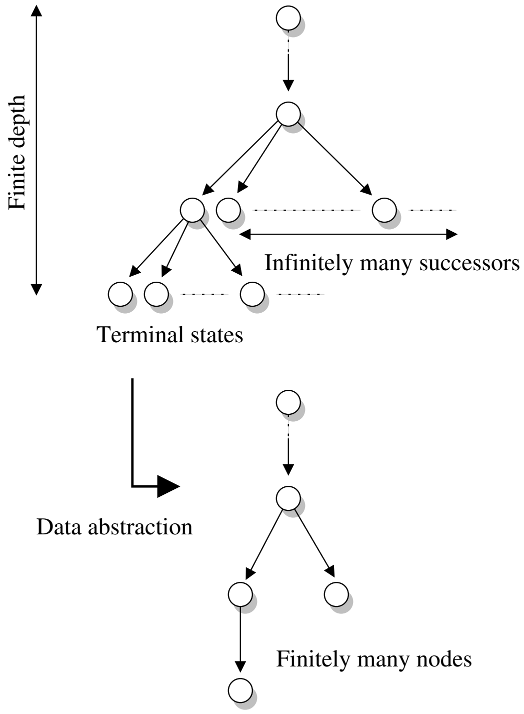

Figure 2: A type I state space and the effect of data abstraction upon it. When the messages are abstracted away the state space shrinks to a finite size.

of type 0. This is unsatisfactory because the capabilities of an intruder need to be artificially restricted, for example by using typing constraints [27], or by limiting the computational capabilities of the intruder, such as the lengths of derivations it uses to generate composed messages [5].

However, there are some results due to Stoller that point to the direction that full automatized analysis of type I models could be possible. He shows that for a fixed number of interleaved runs there exists an a priori upper limit on the amount of computation the intruder must perform in order to attack the protocol, if an attack exists [34, 35]. It could be possible to compute the global resource bounds and then perform finite-state analysis below the bounds. A still better method would be to discover the resource bounds on the fly. This is, conceptually, what our method does.

In the upper part of Fig. 2, a type I infinite state space is shown. When the messages, data, are abstracted away, the state space shrinks to a finite size. This is because the conditions A–C do not hold. In particular, the branching factor in the state graph becomes finite due to that the converse of B is true.

Another way to see our method is that the method performs *selective data abstraction*, abstracting dynamically away such messages and submessages that are not relevant w.r.t. the analysis problem. This is, however, done actually in the reverse way: first *all* data is abstracted away, and then the relevant messages and submessages are 'unabstracted' back. By construction, we demonstrate that the state space obtained in this way can be used to answer the analysis problem correctly, while the space remains finite.

3 Overview of HPA1

In this section we give an overview of HPA1. The next section goes into some technical details. A more thorough exposition can be found from the author's Master's Thesis [14].

3.1 Formal Model

We apply the worst-case threat model as sketched in the introduction. The number of participants is finite. The environment works nondeterministically and all possible interleavings of the participant actions are considered. Intuitively, the environment does not only decide what messages it delivers to the participants, but also decides when the participants execute.

We model cryptography by employing the standard algebraic model, representing cryptographic messages as formal terms. We define a congruence upon the formal terms via a convergent rewrite system [7]: two terms are congruent iff they have the same normal form. The rewrite system we use is parameterizable but otherwise fixed: there is a fixed set of rule templates that can be instantiated for various operators and their combinations. This approach achieves some of the flexibility of a freely chosable rewrite system while keeping the problem solvable.

We define security via user-definable predicates that need to hold in the terminal nodes of a system's state graph. Syntactically, the predicates are words of a formal language that consists of a few atomic predicates that can be combined by using propositional connectives. A terminal state that does not fulfill a given predicate is considered insecure. A protocol instance is considered secure iff there exist no insecure states that can be reached from the initial state of the instance.

3.2 Capabilities and Restrictions

To mention some of the capabilities of our method, it can be said that the method

- supports e.g. multiple symmetric and idempotent symmetric encryption and asymmetric encryption schemes, hash functions, signature systems, tuple constructors and selectors with different arities and revertible transformations;

- does not type messages explicitly, so 'type flaws' can be discovered also;
- does not put any limit on the depth of terms: e.g. multiple encryptions are allowed to an arbitrary level;
- does not restrict the deductive power of the intruder in any way; and
- supports fully non-atomic encryption keys; and
- supports cryptographic protocols where the participants have many optional execution paths via a generic conditional construct.

On the negative side, the method does not currently support associative nor commutative operators. This is the main future challenge, see Sec. 6.

3.3 Analysis Method

The formal model generates a state space for a given initial state. This state space is type I infinite. We reduce the infinite state space to a finite symbolic state space by replacing some parts of the nonsymbolic global states by variables. We then restrict the possible ground values of the variables by giving sets of constraints. To manage the constraints, we employ a constraint solver that simplifies the constraint sets towards certain normal forms. Unfortunately, the details of the solver need to be excluded from our exposition due to space limitations.

3.4 User Interaction

To analyze a protocol, a user describes the protocol using a higher-level procedure language. The user also defines the properties of the cryptographic primitives employed in the protocol and gives the predicates defining secure terminal states. The tool we have implemented compiles the procedural program into the underlying formal model and performs the analysis.

4 Technical Presentation

4.1 Notation and Terminology

We use the following notation and terminology.

If R is a binary relation then R^* denotes its transitive and reflexive closure, i.e. the smallest relation that contains R and is both transitive and reflexive.

Logical implication is denoted by $\Rightarrow$. The set of natural integers $\mathbb{N}$ contains zero.

If f is a function then $f[a \mapsto b]$ is the function defined as

$$f[a \mapsto b](x) = \begin{cases} b & \text{if } x = a \\ f(x) & \text{otherwise.} \end{cases}$$

A finite partial function f is a partial function that is defined only for a finite number of values from the domain of f.

We can extend a function $f : A \to B$ to $f : 2^A \to 2^B$ implicitly, meaning that

$$f(\{x_1, x_2, \dots\}) =_{def} \{f(x_1), f(x_2), \dots\}.$$

4.2 Formal Model

4.2.1 Cryptography

Definition 4.1 (Terms) Let $\mathscr{F}_0, \mathscr{F}_1, \dots$ be collections of symbols, not necessarily mutually disjoint. Let $\mathscr{T}$ be the set of formal terms built from the symbols i.e. the smallest set $\mathscr{T}$ such that

$$t_1, \dots, t_k \in \mathscr{T}, f \in \mathscr{F}_k \Rightarrow f(t_1, \dots, t_k) \in \mathscr{T}.$$

We identify $f()$ with f, i.e. drop the parentheses for symbols of zero arity.

Example 4.1 Let $\mathscr{F}_0 = \{\mathrm{A}, \mathrm{B}\}$ and $\mathscr{F}_2 = \{\mathrm{S}\}$. Then

$$\{\mathrm{A}, \mathrm{S}(\mathrm{A}, \mathrm{A}), \mathrm{S}(\mathrm{S}(\mathrm{A}, \mathrm{B}), \mathrm{S}(\mathrm{B}, \mathrm{B}))\} \subset \mathscr{T}.$$

The set of terms, $\mathscr{T}$, is the formal counterpart of the collection of all the real-world messages that can be sent. In order to model the algebraic properties of the cryptographic primitives, we employ a convergent rewrite system [7] as described next.

Definition 4.2 (Rewrite System $\mathfrak{R}$) Let $P_{1\text{-tuple}}$, $P_{2\text{-tuple}}$, $\dots$, P_{symenc}, P_{pubenc}, P_{signs}, and P_{inv} be given, finite relations on the following domains:

$$\begin{aligned} P_{n\text{-tuple}} &\subseteq \mathscr{F}_n \times (\mathscr{F}_1)^n \\ P_{\text{symenc}} &\subseteq \mathscr{F}_2 \times \mathscr{F}_2 \\ P_{\text{pubenc}} &\subseteq \mathscr{F}_2 \times \mathscr{F}_2 \times \mathscr{F}_1 \times \mathscr{F}_1 \\ P_{\text{signs}} &\subseteq \mathscr{F}_2 \times \mathscr{F}_3 \times \mathscr{F}_1 \times \mathscr{F}_1 \end{aligned}$$

Define $\mathfrak{R}$ to be the rewrite system that is defined by the relations as follows. The following rule templates are instantiated for every combination of operator symbols that appears in the corresponding relation (for example, if $P_{\text{symenc}}(e,d)$ then $d(K, e(K, M))$ rewrites to M for all $K, M \in \mathscr{T}$):

$$\begin{aligned} P_{k\text{-tuple}}(c, s_1, \dots, s_k) \Rightarrow &\quad s_i(c(A_1, \dots, A_k)) \to A_i \\ P_{\text{symenc}}(e,d) \Rightarrow &\quad d(K, e(K, M)) \to M \\ P_{\text{pubenc}}(e,d,p,s) \Rightarrow &\quad d(s(K), e(p(K), M)) \to M \\ P_{\text{signs}}(c,v,p,s) \Rightarrow &\quad v(M, p(K), c(s(K), M)) \to \mathrm{TRUE} \end{aligned}$$

TRUE is the only constant symbol in the templates. In particular, $A_1, \dots, A_k$, K and M are variables ranging over $\mathscr{T}$. We assume and require TRUE to be an element of $\mathscr{F}_0$.

Remark 4.1 (Convergence of $\mathfrak{R}$) $\mathfrak{R}$ is not necessarily convergent. For example, if both $P_{\text{symenc}}(a,b)$ and $P_{\text{symenc}}(b,c)$ (using $R(x_1, \dots, x_k)$ to denote $\langle x_1, \dots, x_k \rangle \in R$), then the term $a(\text{K},b(\text{K},c(\text{K},\text{M})))$ rewrites to both $a(\text{K},\text{M})$ and $c(\text{K},\text{M})$. It is therefore necessary to exclude such combinations of the relations $P_{n\text{-tuple}}, P_{\text{symenc}}, \ldots$ that lead to a nonconvergent rewrite system. We assume that $\mathfrak{R}$ is convergent, i.e. that the set of relations is chosen properly.

Definition 4.3 Assuming $\mathfrak{R}$ is fixed and defined as above, denote the unique normal form (which exists because $\mathfrak{R}$ is convergent) of $t \in \mathscr{T}$ under $\mathfrak{R}$ by $\mathbf{N}(t)$. Denote by $\mathscr{T}_{\mathbf{N}}$ the set $\{t \in \mathscr{T} \mid t = \mathbf{N}(t)\}$, i.e. the set of normal-form terms.

Example 4.2 We have chosen the subscripts of the relation symbols P_x so that they suggest those cryptographic properties the corresponding rewrite rules model. For example, if $P_{\text{symenc}}(\text{E},\text{D})$ and $P_{\text{symenc}}(\text{D},\text{E})$, then E and D can be seen as mutually cancelling symmetric encryption operations, i.e. encryption (E) and decryption (D). For example,

$$\mathbf{N}(\text{E}(\text{K},\text{M})) \neq \mathbf{N}(\text{M})$$

modeling the fact that encryption changes a message to another one, and

$$\mathbf{N}(\text{D}(\text{K},\text{E}(\text{K},\text{M}))) = \mathbf{N}(\text{E}(\text{K},\text{D}(\text{K},\text{M}))) = \mathbf{N}(\text{M}),$$

modeling the mutual cancellation property. In the similar way, $P_{n\text{-tuple}}$ can be used to form tuple constructors and selectors, P_{pubenc} public-key cryptosystem operators and P_{signs} signature operators.

Definition 4.4 (Closure and Rewrite Closure) For $S \subseteq \mathscr{T}$, denote by $\mathbf{C}(S)$ the free term closure of S, i.e. the smallest set S' such that

$$S \subseteq S' \quad \text{and} \quad t_1, \ldots, t_n \in S', f \in \mathscr{F}_n \Rightarrow f(t_1, \ldots, t_n) \in S'.$$

Denote the *rewrite* closure of $M \subseteq \mathscr{M}$ and define it as

$$\mathbf{RC}(M) = \{\mathbf{N}(t) \mid t \in \mathbf{C}(M)\}.$$

Example 4.3 $\mathbf{C}(\emptyset) = \emptyset$.

Example 4.4 $\mathbf{C}(\mathscr{F}_0) = \mathscr{T}$.

Example 4.5 Let $\mathscr{F}_0 = \{\text{A}\}$, $\mathscr{F}_1 = \{\text{S}\}$ and $\mathscr{F}_i = \emptyset$ for $i > 1$. Then $\mathbf{C}(\{a\}) = \{\text{A}, \text{S}(\text{A}), \text{S}(\text{S}(\text{A})), \ldots\}$.

Remark 4.2 Given a set of terms S, $\mathbf{RC}(S)$ denotes the (infinite) collection of terms that can be 'derived' from those in S. This is a common notion in the literature. For example, continuing Example 4.2, we have

$$\text{M} \in \mathbf{RC}(\{\text{E}(\text{K},\text{M}),\text{K}\}),$$

modeling the fact that when both an encrypted message and the corresponding key are known, the encrypted contents can be recovered.

4.2.2 Participants and Environment

We model the participants of cryptographic protocols as state machines that have a finite control state, excluding the cryptographic messages they process, that have a bounded execution length and that have a finite collection of variables where elements of $\mathscr{T}$ can be permanently stored.

Definition 4.5 (Programs) Let $\mathscr{V}$ be a set of *state variables* such that $\mathscr{V} \cap \mathscr{F}_i = \emptyset$ for all i. Let $\mathscr{T}_{\mathscr{V}}$ denote the set of terms that is obtained as $\mathscr{T}$ but by including $\mathscr{V}$ in $\mathscr{F}_0$.[1] The set of *programs* is the smallest set $\mathscr{P}_0$ such that (1) **halt** $\in \mathscr{P}_0$ and (2) for all $p, p' \in \mathscr{P}_0$, $v, v' \in \mathscr{V}$ and $t \in \mathscr{T}_{\mathscr{V}}$:

1. **send** $v.p \in \mathscr{P}_0$,
2. **receive** $v.p \in \mathscr{P}_0$,
3. **generate** $v.p \in \mathscr{P}_0$,
4. **if** $v = v'$ **then** p **else** $p' \in \mathscr{P}_0$, and
5. **let** $[v = t]$ **in** $p \in \mathscr{P}_0$.

Remark 4.3 Intuitively, the programs will have the following semantics: **halt** will denote the act of doing nothing for ever. **send** $v.p$ will denote the act of sending the message contained in the variable v and then continuing according to the program p. Similarly, **recv** will denote receiving a message, storing it to a variable, and **generate** generating a nonce. **if** $\ldots = \ldots$ **then** $\ldots$ **else** $\ldots$ will denote the act of comparing the values of two variables and performing a conditional branch on the basis of the outcome. **let** will denote the act of assigning a value to a variable, where the new value is a term composed of constants and terms already assigned to local state variables.

Definition 4.6 The set of *valid programs* is the set of programs where (1) every variable is assigned in every branch of the program at most once (that is, a variable is never written over); and (2) variables are not referenced before they are given a value. The set of valid programs is denoted by $\mathscr{P}$.

Remark 4.4 The requirement (1) in Def. 4.6 causes no loss of generality.

Definition 4.7 (Local States) A *local state* is a triple $\langle p, \Upsilon, c\rangle$ where $p \in \mathscr{P}$, Υ is a finite partial function $\mathscr{V} \to \mathscr{T}_{\mathbf{N}}$ and $c \in \mathbb{N}$. Denote the set of all local states by S.

Remark 4.5 Intuitively, local states denote single participants in a protocol. p is the program a participant is running, Υ denotes the contents of its state variables, and c is a nonce counter that is included in the local state so that we can later create distinct nonce objects when necessary.

Notation 4.1 We extend the domain of Υ from $\mathscr{V}$ to $\mathscr{T}_{\mathscr{V}}$ in the natural way:

$$\Upsilon(f(t_1, \ldots, t_k)) =_{def} f(\Upsilon(t_1), \ldots, \Upsilon(t_k)).$$

Definition 4.8 (Global States) Let $\mathcal{N}$ be a finite (fixed) set of *participant names*. ($|\mathcal{N}|$ will be the number of interleavedly executing participants.) A *global state* is a pair $\langle \mathscr{L}, M\rangle$ where $\mathscr{L}$ is a total function $\mathcal{N} \to S$ and $M \subset \mathscr{T}_{\mathbf{N}}$, M finite. Let $\mathscr{G}$ denote the set of all global states.

[1] That is, abusing the notation a bit, $\mathscr{T}_{\mathscr{V}} = \mathbf{C}(\mathscr{F}_0 \cup \mathscr{V})$.

Remark 4.6 Intuitively, a global state includes the knowledge of the environment, modeled as the set of terms M, and the local states of the protocol participants.

The transition relation between global states—introduced next—defines the collective behaviour for a system consisting of the environment and a set of participants.

Definition 4.9 (Transition Relation) Define the family of labeled transition relations $\underset{\ell}{\rightarrow}$ over $\mathscr{G}^2$ as follows: $\langle \mathscr{L}, M\rangle \underset{\ell}{\rightarrow} \langle \mathscr{L}', M'\rangle$ iff $\mathscr{L}' = \mathscr{L}[n \mapsto \langle p', \Upsilon', c'\rangle]$ for some $n \in \mathcal{N}$, $\mathscr{L}(n) = \langle p, \Upsilon, c\rangle$, $\ell = n.p$ (the label), and one of the following holds:

1. $p = \textbf{send}\ v.p'$, $\Upsilon' = \Upsilon$, $M' = M \cup \Upsilon(v)$ and $c' = c$.
2. $p = \textbf{receive}\ v.p'$, $\Upsilon' = \Upsilon[v \mapsto m]$ for some m such that $m \in \mathbf{RC}(M)$, $M' = M$ and $c' = c$.
3. $p = \textbf{generate}\ v.p'$, $\Upsilon' = \Upsilon[v \mapsto \mathrm{nonce}^n_c]$, $M' = M$ and $c' = c + 1$.
4. $p = \textbf{let}\ [v = t]\ \textbf{in}\ p'$, $\Upsilon' = \Upsilon[v \mapsto \mathbf{N}(\Upsilon(t))]$, $M' = M$ and $c' = c$.
5. $p = \textbf{if}\ v = v'\ \textbf{then}\ p'\ \textbf{else}\ x$, $\Upsilon(v) = \Upsilon(v')$, $\Upsilon' = \Upsilon$, $M' = M$ and $c' = c$.
6. $p = \textbf{if}\ v = v'\ \textbf{then}\ x\ \textbf{else}\ p'$, $\Upsilon(v) \neq \Upsilon(v')$, $\Upsilon' = \Upsilon$, $M' = M$ and $c' = c$.

Denote the union $\bigcup_\ell \underset{\ell}{\rightarrow}$ by $\rightarrow$. The set $\{g \in \mathscr{G} \mid g \rightarrow^* g' \Rightarrow g = g'\}$ is the set of *terminal states* and is denoted by $\mathscr{G}_T$.

We assume and require $\mathrm{nonce}^n_k \in \mathscr{F}_0$ for all $n \in \mathcal{N}, k \in \mathbb{N}$.

Remark 4.7 Because programs have finite lengths, for every global state $g \in \mathscr{G}$ the trail $g \rightarrow g_1 \rightarrow g_2 \rightarrow \cdots$ is of finite length.

Remark 4.8 The control state of a participant $\langle p, \upsilon, c\rangle$ can now seen to be p. Thus the set of distinct control states for a given participant is definitely finite because all programs are finite, containing thus only a finite number of distinct subprograms.

4.3 Analysis Problem

We aim for solving the following problem: given a predicate T over $\mathscr{G}_T$ and a global state $g \in \mathscr{G}$, does there exist $g' \in \mathscr{G}_T$ such that $g \rightarrow^* g'$ and $\neg T(g')$? In other words: does there exist a reachable terminal state such that the given predicate T is not fulfilled in it?

Because we are analyzing the security of cryptographic protocols, T can be seen to be a predicate defining security. In essence, T should be true in exactly those terminal states that are considered safe and harmless.

The following remark is perhaps due. Some authors prefer to use the *correspondence model* of security, where security is defined not in terms of the terminal states alone, but instead as a property of the sequences of actions leading to the terminal states. We have two reasons for not adopting this idea. First, when only terminal states need to be considered, advanced state space reduction methods (e.g. removing unnecessary interleavings) can be more freely used. Second, our opinion is that the security of a protocol is ultimately a function of the terminal states only. If there is an insecure state before the action has ended, then there must be at least one insecure terminal state also. Otherwise the notion of security is not sound.

4.3.1 Security Predicates

It is conceivable that some predicates over $\mathscr{G}_T$ can be very hard to analyze. We therefore limit us to the set of predicates defined as follows.

Definition 4.10 Let $\mathscr{V}_\mathcal{N} = \mathscr{V} \times \mathcal{N}$ be the set of *qualified state variables*. An element $\langle v, n\rangle \in \mathscr{V}_\mathcal{N}$ will be written as $n : v$. Let $\mathscr{T}_Q$ be the set of terms obtained as $\mathscr{T}$ but by including $\mathscr{V}_\mathcal{N}$ in $\mathscr{F}_0$. We define the function $\tilde{\Upsilon} : \mathscr{G} \times \mathscr{T}_Q \rightarrow \mathscr{T}_\mathbf{N}$ as

$$\tilde{\Upsilon}(g,t) = \begin{cases} \mathbf{N}(t) & \text{if } t \in \mathscr{T} \\ \Upsilon(t) & \text{if } t = n : v,\ g = \langle \mathscr{L}, M\rangle,\ \mathscr{L}(n) = \langle p, \Upsilon, c\rangle \\ f(\tilde{\Upsilon}(g,t_1), \ldots, \tilde{\Upsilon}(g,t_k)) & \text{if } t = f(t_1, \ldots, t_k). \end{cases}$$

Remark 4.9 Qualified state variables are used to refer to the state of individual participants of the protocol. Obviously, $n : v$ denotes the state variable v of the participant n. $\tilde{\Upsilon}$ 'evaluates' a term that contains qualified state variables, exactly in the same way as the valuation functions Υ give ground values for terms containing ordinary state variables.

Definition 4.11 Define the language Φ as the smallest language such that $\underline{\top} \in \Phi$, and for all $\phi_1, \phi_2 \in \Phi$ and $t_1, t_2 \in \mathscr{T}_Q$,

$$\underline{\neg \phi_1}, \quad \underline{(\phi_1 \wedge \phi_2)}, \quad \underline{\mathrm{secret}(t_1)}, \quad \underline{t_1 = t_2} \quad \in \Phi.$$

(We underline the words of Φ in order to make a clear distinction between logical notation in general and the language Φ.)

The elements of Φ are exactly the syntactical presentations of the predicates we allow. Semantics are defined as follows.

Definition 4.12 Read $g \models \phi$ for $g \in \mathscr{G}_T$ and $\phi \in \Phi$ as 'ϕ holds in the terminal state g'. Denote $g \not\models \phi$ iff $g \models \phi$ does not hold. Define:

$$\begin{array}{lll} g \models \underline{\top} & \text{for all } g & \\ g \models \underline{\neg\phi} & \text{iff} & g \not\models \underline{\phi} \\ g \models \underline{(\phi_1 \wedge \phi_2)} & \text{iff} & g \models \underline{\phi_1} \quad \text{and} \quad g \models \underline{\phi_2} \\ g \models \underline{t_1 = t_2} & \text{iff} & \tilde{\Upsilon}(g,t_1) = \tilde{\Upsilon}(g,t_2) \\ g \models \underline{\mathrm{secret}(t)} & \text{iff} & \tilde{\Upsilon}(g,t) \notin \mathbf{RC}(M) \text{ when } g = \langle \mathscr{L}, M\rangle \end{array}$$

Example 4.6 Let A and B be two participants that both have a state variable V. Then the predicate

$$\underline{(\mathrm{secret}(A:V) \wedge A:V = B:V)}$$

holds in exactly those terminal states in which the values of the state variable V agree for the two participants, and the common value (a term) is not known to the environment.

4.3.2 *On Solving the Analysis Problem*

A standard solution for solving the problem we presented would be exhaustive state space enumeration. However, the number of states reachable from $g \in \mathscr{G}$ is infinite whenever there exists at least one **recv** action in the programs of the participants in g (unless the set of messages known to the environment is totally empty, but we can assume that not to be the case). Therefore standard state space enumeration is not applicable. Instead, symbolic state space enumeration must be used. This is our next subject.

4.4 Symbolic Representation

Definition 4.13 (Substitutions) Let X be a set of variables. Denote by $\mathscr{T}_X$ the set of terms obtained as $\mathscr{T}$ but by including X in $\mathscr{F}_0$. A mapping $\sigma : X \to \mathscr{T}_X$ ($\sigma : X \to \mathscr{T}$) is a *substitution* (*ground substitution*).

Notation 4.2 If σ is a substitution we extend its domain implicitly from X to $\mathscr{T}_X$ in the natural way:

$$\sigma(f(t_1,\dots,t_k)) =_{def} f(\sigma(t_1),\dots,\sigma(t_k)).$$

4.4.1 *Symbolic States*

Definition 4.14 (Constraint Sets) A *constraint set* is a finite set of the following elements:

1. $\mathrm{Eq}(t,t')$ for $t,t' \in \mathscr{T}_X$;
2. $\mathrm{Ineq}(t,t')$ for $t,t' \in \mathscr{T}_X$; and
3. $\mathrm{InClos}(t,M)$ for $t \in \mathscr{T}_X$ and $M \subset \mathscr{T}_X$, M finite.

Denote the collection of all constraint sets by $\mathscr{C}$. If σ is a substitution we extend its domain implicitly to contain $\mathscr{C}$ by defining:

$$\begin{aligned}
\sigma(\emptyset) &=_{def} \emptyset \\
\sigma(\{\mathrm{Eq}(t,t'),\dots\}) &=_{def} \{\mathrm{Eq}(\sigma(t),\sigma(t'))\} \cup \sigma(\{\dots\}) \\
\sigma(\{\mathrm{Ineq}(t,t'),\dots\}) &=_{def} \{\mathrm{Ineq}(\sigma(t),\sigma(t'))\} \cup \sigma(\{\dots\}) \\
\sigma(\{\mathrm{InClos}(t,M),\dots\}) &=_{def} \{\mathrm{InClos}(\sigma(t),\sigma(M))\} \cup \sigma(\{\dots\}),
\end{aligned}$$

i.e. by applying the substitution to all the terms contained in a constraint set.

Definition 4.15 (Ground Constraint Sets) A *ground constraint set* is a constraint set where no elements of X appear, i.e a set $C \in \mathscr{C}$ for which $C = \sigma(C)$ for all substitutions $\sigma : X \to \mathscr{T}_X$. A ground constraint set C is *sound* iff the following hold:

1. $\mathrm{Eq}(t,t') \in C$ implies that $t = t'$ and $t,t' \in \mathscr{T}_{\mathbf{N}}$;
2. $\mathrm{Ineq}(t,t') \in C$ implies that $t \neq t'$ and $t,t' \in \mathscr{T}_{\mathbf{N}}$; and
3. $\mathrm{InClos}(t,M) \in C$ implies that $t \in \mathbf{RC}(M)$, (which implies $t \in \mathscr{T}_{\mathbf{N}}$), and $M \subset \mathscr{T}_{\mathbf{N}}$.

Definition 4.16 (Symbolic States) A *symbolic global state* is a triple $\langle \hat{\mathscr{L}}, \hat{M}, C\rangle$ where $C \in \mathscr{C}$, $\hat{M} \subset \mathscr{T}_X$ and $\hat{M}$ is finite, and $\hat{\mathscr{L}}$ is a total function from $\mathcal{N}$ to *symbolic local states*. A symbolic local state is a triple $\langle p, \hat{\Upsilon}, c\rangle$ where $p \in \mathscr{P}$ and $c \in \mathbb{N}$ are as for nonsymbolic local states and $\hat{\Upsilon}$ is a finite partial function $\mathscr{V} \to \mathscr{T}_X$. Denote the set of all symbolic global states by $\hat{\mathscr{G}}$.

If σ is a substitution we extend its domain implicitly to contain $\hat{\mathscr{G}}$ by defining

$$\sigma(\langle \hat{\mathscr{L}}, \hat{M}, C\rangle) =_{def} \langle \sigma(\hat{\mathscr{L}}), \sigma(\hat{M}), \sigma(C)\rangle$$

where $\sigma(\hat{\mathscr{L}})$ is defined as

$$\hat{\mathscr{L}}(n) = \langle p, \hat{\Upsilon}, c\rangle \Rightarrow \sigma(\hat{\mathscr{L}})(n) =_{def} \langle p, \sigma(\hat{\Upsilon}), c\rangle$$

($n \in \mathcal{N}$), and

$$\hat{\Upsilon}(v) = t \Rightarrow \sigma(\hat{\Upsilon})(v) =_{def} \sigma(t)$$

($v \in \mathscr{V}$, and understanding that if $\hat{\Upsilon}(v)$ is not defined then nor is $\sigma(\hat{\Upsilon})(v)$).

Definition 4.17 (Solutions) Let C be a constraint set. A ground substitution σ is *a solution to* C iff $\sigma(C)$ is sound. The set of solutions to C is denoted by $\mathrm{sols}(C)$.

Definition 4.18 A symbolic global state $\hat{g} = \langle \hat{\mathscr{L}}, \hat{M}, C\rangle$ *represents* exactly those global states $\langle \sigma(\hat{\mathscr{L}}), \sigma(\hat{M})\rangle$ that are obtained by choosing σ from $\mathrm{sols}(C)$. We denote this set of global states by $\rho(\hat{g})$. More rigorously:

$$\rho(\langle \hat{\mathscr{L}}, \hat{M}, C\rangle) = \{\langle \sigma(\hat{\mathscr{L}}), \sigma(\hat{M})\rangle \mid \sigma \in \mathrm{sols}(C)\}.$$

4.4.2 *Symbolic Transition Relation*

We shall next define a symbolic transition relation, a symbolic counterpart of the transition relation $\to$ presented in Def. 4.9. The relation between the symbolic and nonsymbolic transition relations will be presented after the definitions have been given.

Definition 4.19 Define the function $\mathrm{succ}_1 : \hat{\mathscr{G}} \times \mathcal{N} \mapsto 2^{\hat{\mathscr{G}}}$ as follows. Let $\hat{g} \in \hat{\mathscr{G}} = \langle \hat{\mathscr{L}}, \hat{M}, C\rangle$ and let $n \in \mathcal{N}$. Let

$\hat{\mathscr{L}}(n) = \langle p, \hat{\Upsilon}, c\rangle$. Then $\mathrm{succ}_1(\hat{g}, n)$ is defined as:

$$\mathrm{succ}_1(\hat{g}, n) =$$

$$\begin{cases} \emptyset & \text{if } p = \textbf{halt} \\ \{\langle \hat{\mathscr{L}}_1, \hat{M} \cup \{\hat{\Upsilon}(v)\}, C\rangle\} & \text{if } p = \textbf{send } v.p' \\ \{\langle \hat{\mathscr{L}}_2, \hat{M}, C \cup \{\mathrm{InClos}(x, \hat{M})\}\rangle\} & \text{if } p = \textbf{receive } v.p' \\ \quad \text{where } x \in X \text{ does not appear in } \hat{g} & \\ \{\langle \hat{\mathscr{L}}_3, \hat{M}, C\rangle\} & \text{if } p = \textbf{generate } v.p' \\ \{\langle \hat{\mathscr{L}}_4, \hat{M}, C\rangle\} & \text{if } p = \textbf{let } [v = t] \textbf{ in } p' \\ \{\langle \hat{\mathscr{L}}_5, \hat{M}, C \cup \{\mathrm{Eq}(\hat{\Upsilon}(v), \hat{\Upsilon}(v'))\rangle, & \\ \langle \hat{\mathscr{L}}_6, \hat{M}, C \cup \{\mathrm{Ineq}(\hat{\Upsilon}(v), \hat{\Upsilon}(v'))\rangle\} & \\ & \text{if } p = \textbf{if } v = v' \textbf{ then } p' \textbf{ else } p'' \end{cases}$$

where $\hat{\mathscr{L}}_1, \ldots, \hat{\mathscr{L}}_6$ are given by

$$\begin{aligned} \hat{\mathscr{L}}_1 &= \hat{\mathscr{L}}[n \mapsto \langle p', \hat{\Upsilon}, c\rangle] \\ \hat{\mathscr{L}}_2 &= \hat{\mathscr{L}}[n \mapsto \langle p', \hat{\Upsilon}[v \mapsto x], c\rangle] \\ \hat{\mathscr{L}}_3 &= \hat{\mathscr{L}}[n \mapsto \langle p', \hat{\Upsilon}[v \mapsto \mathrm{nonce}_c^n], c+1\rangle] \\ \hat{\mathscr{L}}_4 &= \hat{\mathscr{L}}[n \mapsto p', \hat{\Upsilon}[v \mapsto \hat{\Upsilon}(t)], c\rangle] \\ \hat{\mathscr{L}}_5 &= \hat{\mathscr{L}}[n \mapsto \langle p', \hat{\Upsilon}, c\rangle] \\ \hat{\mathscr{L}}_6 &= \hat{\mathscr{L}}[n \mapsto \langle p'', \hat{\Upsilon}, c\rangle]. \end{aligned}$$

Definition 4.20 We define the family of labeled symbolic transition relations $\xrightarrow[\ell]{s}$ over $\hat{\mathscr{G}}^2$ as follows: $\hat{g} \xrightarrow[\ell]{s} \hat{g}'$ iff $\hat{g} = \langle \hat{\mathscr{L}}, \hat{M}, C\rangle$, $n \in \mathcal{N}$, $\hat{\mathscr{L}}(n) = \langle p, \hat{\Upsilon}, c\rangle$, $\hat{g}' \in \mathrm{succ}_1(\hat{g}, n)$ and $\ell = n.p$. Denote the union $\bigcup_\ell \xrightarrow[\ell]{s}$ by $\xrightarrow{s}$.

The following proposition shows the relation between $\rightarrow$ and $\xrightarrow{s}$:

Proposition 4.1 *Let $\hat{g}$ be a symbolic state. Then*

$$\bigcup_{g \in \rho(\hat{g})} \{g' \mid g \xrightarrow[\ell]{} g'\} = \bigcup_{\hat{g}' : \hat{g} \xrightarrow[\ell]{s} \hat{g}'} \rho(\hat{g}').$$

(Proof sketch) Prove in cases of p (for $\ell = n.p$) on the basis of the previous definitions. □

Corollary 4.2 *Let $g \in \mathscr{G}$ and $\hat{g} \in \hat{\mathscr{G}}$ such that $\rho(\hat{g}) = \{g\}$. Then*

1. *If $g \rightarrow^* g'$ then $\exists \hat{g}' : \hat{g} \xrightarrow{s}{}^* \hat{g}'$ such that $g' \in \rho(\hat{g}')$.*
2. *If $\hat{g} \xrightarrow{s}{}^* \hat{g}'$ and $g' \in \rho(\hat{g}')$, then $g \rightarrow^* g'$.*

Combined with the fact that $\hat{g} \in \hat{\mathscr{G}}$ the set implies that

$$\{\hat{g}' \mid \hat{g} \xrightarrow{s} \hat{g}'\}$$

is finite (a consequence of Defs. 4.19 and 4.20), the Corollary 4.2 gives a sound and complete method for enumerating symbolically the whole state space, assuming that it is possible to decide for all the terminal symbolic states $\hat{g}$ generated if

$$\overset{?}{\exists} g \in \rho(\hat{g}) : g \not\models \phi$$

for any given $\phi \in \Phi$.

The method we use for deciding the problem above is the following: first, the given symbolic terminal state $\hat{g}$ and the predicate ϕ are collectively mapped into a set of constraints C. The new set C is merged with the constraint set of $\hat{g}$, obtaining a new symbolic state $\hat{g}'$. C is chosen so that $\rho(\hat{g}')$ contains exactly those states from the set $\rho(\hat{g})$ for which ϕ does not hold. Thus it is enough to check whether or not

$$\rho(\hat{g}') \overset{?}{=} \emptyset.$$

For the first half (generating the auxiliary constraint set) we trust in the intuition of the reader and just mention that there exists an algorithm for constructing C. We will discuss the second half, deciding whether or not $\rho(\hat{g}')$ is empty, next.

4.4.3 Managing Symbolic States

Definition 4.21 Let $G \subseteq \hat{\mathscr{G}}$. A function $f : G \mapsto 2^{\hat{\mathscr{G}}}$ is *a sound one-to-many transformation for* G iff for all $\hat{g} \in G$,

$$\rho(\hat{g}) = \bigcup_{\hat{g}' \in f(\hat{g})} \rho(\hat{g}'),$$

and $f(\hat{g})$ is finite.

As a part of our method we have developed a sound one-to-many transformation Z for a certain G with the following extra properties: there exists a partial ordering $\sqsubseteq$ over $\hat{\mathscr{G}}$ such that

(i) $\hat{g} \in Z(\hat{g})$ implies that $Z(\hat{g}) = \hat{g}$, $\hat{g}$ is a minimal element w.r.t. $\sqsubseteq$ and that $\rho(\hat{g}) \neq \emptyset$;

(ii) $Z(\hat{g}) = \{\hat{g}_1, \ldots, \hat{g}_k\}$ implies that $\hat{g}_i \sqsubseteq \hat{g}$ for all i;

(iii) $Z(\hat{g}) = \emptyset$ for all minimal $\hat{g}$ such that $\rho(\hat{g}) = \emptyset$;

(iv) all descending chains $\hat{g} \sqsupset \hat{g}' \sqsupset \cdots$ are finite;

(v) G is closed under Z, i.e. $\hat{g} \in G$ implies $Z(\hat{g}) \subset G$;

(vi) G is closed under $\xrightarrow{s}$; and

(vii) all ground symbolic states, i.e. symbolic states where no elements of X do appear, belong to G.

Definition 4.22 Given the particular one-to-many transformation Z presented previously, we define the function $Z_m : 2^{\hat{\mathscr{G}}} \mapsto 2^{\hat{\mathscr{G}}}$ as

$$Z_m(A) = \bigcup_{\hat{g} \in A} Z(\hat{g}).$$

Define then

$$Z_{\mathrm{fix}}(A) = \begin{cases} A & \text{if } A = Z_m(A) \\ Z_{\mathrm{fix}}(Z_m(A)) & \text{otherwise.} \end{cases}$$

The value of Z_{fix} for a finite $A \subset G$ is well-defined due to the properties of Z; a fixpoint always exists.

Proposition 4.3

$$Z_{\text{fix}}(\hat{g}) = \emptyset \quad \textit{iff} \quad \rho(\hat{g}) = \emptyset.$$

(Proof sketch) ($\Rightarrow$) Because Z is a sound one-to-many transformation. ($\Leftarrow$) All the symbolic states $\hat{g}'$ in $Z_{\text{fix}}(\hat{g})$ are minimal w.r.t. to $\sqsubseteq$ and have $\rho(\hat{g}') \neq \emptyset$ because Z_{fix} is a fixpoint of Z_m. □

4.4.4 *On the Implementation of Z*

We do not have room for discussing the transformation Z in detail, but there is a major point that we would like to mention.

Typically, systems that involve equational reasoning in the presence of a rewrite system, employ the narrowing algorithm or some other, similar algorithms, for solving the equations. [7] In contrast to this, we do not use narrowing nor any of its relatives, at least not explicitly. Instead of this, the whole solving process that is required for implementing Z is intertwined with the analysis process in a way that removes the need for generic equational reasoning.

This is possible because the equations that emerge during forward search have a very special form that can be exploited. The most important property is that any variable that appears in the equations and terms is *bounded* in the sense that it denotes a part of a message that has been originally sent by the environment. That is, it is a part of a message that *has been known to the environment at some earlier step*.

For this and some other reasons, it seems that full-fledged narrowing is not needed.

4.5 Actual State Space Enumeration

We are now ready to present a rudimentary but complete symbolic state space enumeration algorithm for our problem. The input to the algorithm is a global state $g \in \mathscr{G}$ and a predicate $\phi \in \Phi$. The algorithm, shown in Fig. 3, decides whether or not

$$\overset{?}{\exists} g' \in \mathscr{G} : g \rightarrow^* g' \wedge g' \not\models \phi.$$

This rudimentary algorithm can be enhanced in various ways. We mention some methods, which are included in the full method of ours.

4.5.1 *Early Pruning of Empty Symbolic States*

If $\rho(\hat{g}) = \emptyset$ and $\hat{g} \overset{s}{\rightarrow}^* \hat{g}'$, then also $\rho(\hat{g}') = \emptyset$. Thus, some parts of the symbolic space can be pruned early by checking that $Z_{\text{fix}}(\hat{g}) \neq \emptyset$ for all those states that are added to the working set W on the line 15 of the basic algorithm. Obviously, it is waste of resources to calculate the whole $Z_{\text{fix}}(\hat{g})$ just to check whether it is empty or not. Therefore the line 15 can be rewritten as:

$W \leftarrow W \cup Z_{\text{fix}}(S)$.

1: Denote $g = \langle \mathscr{L}, M \rangle$
2: $W \leftarrow \{\langle \mathscr{L}, M, \emptyset \rangle\}$
3: **while** $W \neq \emptyset$ **do**
4:    Choose (arbitrarily) $\hat{g} \in W$
5:    $W \leftarrow W \setminus \{\hat{g}\}$
6:    $S \leftarrow \{\hat{g}' \mid \hat{g} \overset{s}{\rightarrow} \hat{g}'\}$
7:    **if** $S = \emptyset$ **then** *; $\hat{g}$ is terminal*
8:       Denote $\hat{g} = \langle \hat{\mathscr{L}}, \hat{M}, C \rangle$
9:       Map $\hat{g}$ and ϕ to a constraint set C'
10:       $\hat{g}' \leftarrow \langle \hat{\mathscr{L}}, \hat{M}, C \cup C' \rangle$
11:       **if** $Z_{\text{fix}}(\hat{g}') \neq \emptyset$ **then**
12:          Insecure state found. STOP.
13:       **end if**
14:    **else**
15:       $W \leftarrow W \cup S$
16:    **end if**
17: **end while**

Figure 3: The rudimentary symbolic state space enumeration and model checking algorithm.

This incorporates both the early pruning idea and the memoization of the intermediate results of Z_{fix}.

4.5.2 *Removal of Redundant Interleavings*

Typically, there are many traces $g \rightarrow g_1 \rightarrow \cdots \rightarrow g_k \rightarrow g'$ that lead from g to g'. The different traces typically correspond to different interleavings of the internal actions of the local participants. Most of these interleavings are redundant. The redundant interleavings can be eliminated by using the concept of a *step transition* that abstracts away some of the interleavings.

We use the following definition of a step: A trace $g_1 \xrightarrow[n_1 \cdot p_1]{} \cdots \xrightarrow[n_{k-1} \cdot p_{k-1}]{} g_k$ ($k \geq 1$) is a *step* iff (1) $n_i = n_j$ for all $1 \leq i, j < k$; and (2) for all $1 \leq i < k$, if $p_i = \textbf{receive}\ v.p'$, $g_i = \langle \mathscr{L}, M \rangle$ and $g_1 = \langle \mathscr{L}', M' \rangle$, then $\mathbf{RC}(M) = \mathbf{RC}(M')$.

Intuitively, a single participant is allowed to execute as long as it does not try to receive a message after it has itself sent a message to the environment that has genuinely increased its knowledge.

We denote by $\rightsquigarrow$ the *maximal step transition relation*, i.e. $g \rightsquigarrow g'$ iff there exists a trace $g \rightarrow \cdots \rightarrow g'$ that is a step and that is not properly contained in any other step starting from g. It can be shown that if $g \in \mathscr{G}$ and $g' \in \mathscr{G}_T$ then $g \rightsquigarrow^* g'$ iff $g \rightarrow^* g'$.

We are able to define also a *maximal symbolic step transition relation* over $\hat{\mathscr{G}}^2$, the symbolic counterpart of $\rightsquigarrow$, which is the actually used relation. However, the theory is beyond the scope of this paper.

4.5.3 Security Predicate Rewriting and Early Pruning

For some choices of $\phi \in \Phi$ and $g \in \mathscr{G}$, it can be decided immediately whether or not there exists a terminal state $g' \in \mathscr{G}_T : g \to^* g'$ for which $g' \not\models \phi$. In such cases the search can be immediately pruned. This idea can be reflected to the symbolic model. As an additional optimization, the predicate ϕ can be rewritten on the fly, simplifying the predicate using the standard rules for propositional calculus after those subformulae in ϕ that can be already evaluated have been replaced with their truth values.

4.5.4 Symbolic State Subsumption and Symmetries

Some symbolic states *subsume* some others in the sense that for two symbolic states $\hat{g}$ and $\hat{g}'$, $\rho(\hat{g}) \subseteq \rho(\hat{g}')$. If both symbolic states are encountered during search the search can be obviously pruned at $\hat{g}$. This idea can be extended in various ways. We have specified a combined symmetry/subsumption method that combines an incomplete subsumption check with symmetry method that considers such states equivalent that can be obtained from each others by changing the names of different protocol runs. These methods require further research.

4.6 Completeness of the Algorithm

The algorithm we have sketched is complete for the class of protocols it is defined for. The completeness follows basically from two facts: (1) the generated terminal symbolic global states contain exactly those nonsymbolic global states that are reachable from the initial state; and (2) the algorithm for detecting whether a symbolic global state denotes a state for which the given security predicate is false is sound and complete.

4.7 Efficiency

The title of this paper suggests that we consider the algorithm to be an efficient one. At this point, we claim rather categorial than CPU-time efficiency because our implementation (discussed in the next section) is rather unoptimized. Nevertheless, analysis time seems to be on par with the existing finite-state methods. However, while the existing methods analyze only a finite state space, we analyze the whole infinite one. This should be kept in mind.

More importantly, the method is efficienct in the sense that large unimportant clusters in the underlying state space are combined and analyzed as single units. That is, the method interacts with the dynamics of the protocol instance. A straightforward finite-state method needs to guess *a priori* the messages the intruder tries to send to the participants, and then check the effects of such messages. In contrast to this, our method chooses the messages delayedly, using all available information to guide the selection.

5 Practical Implementation

We have implemented our algorithms as an end-user tool. The user of the tool describes the protocol in a higher-level procedural language that is compiled into the underlying formal model. The security predicates are also given by the user. They can be associated with particular end conditions of individual participants, giving enhanced flexibility in the protocol specification (this does not change the formal expressive power of the method). The actual analysis process is fully automatic and does not require user interaction.

In Appendix A, we present as an example code for the well-known Needham-Schröder Public-Key Protocol [29] that contains a security problem first identified by Lowe [17]. The code can be seen to consist of five parts.

First, the six operators used in the listing are declared. Next, their mutual algebraic properties are given. The `declare-atoms` clause lists in advance the atoms that will appear in the program listing. The initial knowledge of the intruder is defined by using the `declare-public` forms.

After the declarations, four procedures are defined. `get-my-key` and `get-her-key` are procedures that fetch the private or public key of the given participant. `ns-initiator` and `ns-responder` give models for the two rôles of the protocol.

The security predicates to check are specified using the `halt-check` form. In this example there is only one such a form. It is found from the end of the procedure `ns-initiator`. The semantics are that the predicate after the `halt-check` keyword must hold whenever the participant has ended its executing at the same `halt-check` statement. The predicate here can be transliterated as:

> "There may not exist a participant Z such that the participant is not ourself, that it believes it has been communicating with us, that it has terminated succesfully but that it is not the participant we believe we have been communicating with."

This predicate does not work well if there are multiple initiators and responders specified; actually the nonces should be compared instead of the less exact identities. Our main purpose here is however only pedagogical demonstration.

Finally, an initiator and a responder are instantiated. The search can begin.

The analyzer is able to find the problem and gives the output shown in Fig. 3 as a trace that leads to an insecure state.

```
[alice] receives igor
[alice] sends rsa-enc(rsa-dpub(igor-keym), cons(N2.0, alice))
[bob] receives rsa-enc(rsa-dpub(bob-keym), cons(N2.0, alice))
[bob] sends rsa-enc(rsa-dpub(alice-keym), cons(N2.0, N1.0))
[alice] receives rsa-enc(rsa-dpub(alice-keym), cons(N2.0, N1.0))
[alice] sends rsa-enc(rsa-dpub(igor-keym), N1.0)
[bob] receives rsa-enc(rsa-dpub(bob-keym), N1.0)
```

Figure 4: Example output for the Needham-Schröder Public-Key Protocol.

6 Future Work

We see that the major future challenge is the inclusion of support for associative and commutative operators in the method. These operators include e.g. exclusive-or (associative, commutative and idempontent), arithmetic addition (associative and commutative), associative concatenation (associative), commutative encryption and the Diffie-Hellman modular exponentation.

7 Conclusions

In this paper we have presented a new method for the automatic analysis and verification of security protocols. What is new is that the method is able to analyze the whole infinite state space generated by a bounded number of participants without requiring user interaction. Morever, the method is efficient in the sense that large state sets can be combined and handled as units. The dynamical behaviour of the protocol system under analysis controls the analysis process, and thus for example encrypted messages that will not be accepted by the receiving party are not explicitly generated at all.

This method is to our knowing the first method that performs infinite-state analysis efficiently, completely and without user interaction for cryptographic protocols.

Acknowledgements. Thanks are due to Catherine Meadows, Tuomas Aura, Pekka Nikander, Tommi Junttila and Mika Kojo, and the anonymous referees of the extended abstract.

References

1. Boyd, C. A formal framework for authentication. In *Proceedings of the Second European Symposium on Research in Computer Security — ESORICS 92* (1992), pp. 273–292.
2. Bryant, R. E. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys 24*, 3 (Sept. 1992), 293–318.
3. Burrows, M., Abadi, M., and Needham, R. A logic of authentication. In *Proceedings of the Royal Society, Series A, 426, 1871* (Dec. 1989). Appeared also as SRC Research Report 39 (February 1989) and, in a shortened form, in ACM Transactions on Computer Systems 8(1):18–36 (February 1990).
4. ———. The scope of a logic of authentication. In *Distributed Computing and Cryptography: Proceedings of a DIMACS Workshop* (Oct. 1989), pp. 119–126.
5. Clarke, E. M., Jha, S., and Marrero, W. Using state space exploration and a natural deduction style message derivation engine to verify security protocols. In *Proceedings of the IFIP Working Conference on Programming Concepts and Methods (PROCOMET)* (1998).
6. Dang, Z., and Kemmerer, R. Using the ASTRAL model checker for cryptographic protocol analysis. In *Proceedings of the DIMACS Workshop on Design and Formal Verification of Security Protocols* (Sept. 1997). The proceedings is available only in electronic form.
7. Dershowitz, N., and Jouannaud, J.-P. *Handbook of Theoretical Computer Science*, vol. B. Elsevier Science Publishers B.V., 1990, ch. Rewrite Systems, pp. 243–320.
8. Dolev, D., and Yao, A. C. On the security of public key protocols. *IEEE Transactions on Information Theory 29*, 2 (Mar. 1983), 198–208.
9. Formal Systems (Europe) Ltd. Failures-divergence refinement — FDR 2 — user manual, 1997. Available via URL `ftp://ftp.comlab.ox.ac.uk/pub/Packages/FDR`.
10. Germeau, F., and Leduc, G. Model-based design and verification of security protocols using LOTOS. In *Proceedings of the DIMACS Workshop on Design and Formal Verification of Security Protocols* (Sept. 1997). The proceedings is available only in electronic form.
11. Gong, L., Needham, R., and Yahalom, R. Reasoning about belief in cryptographic protocols. In *Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy* (May 1990), IEEE Computer Society Press, pp. 234–248.
12. Gritzalis, S., Nikitakos, N., and Georgiadis, P. Formal methods for the analysis and design of cryptographic protocols: A state-of-the-art review. In *Third IFIP TC6/TC11 Working Conference on Communications and Multimedia Security* (1997), pp. 119–132.
13. Hoare, C. A. R. *Communicating Sequential Processes*. Prentice-Hall, 1985.
14. Huima, A. Analysis of cryptographic protocols via symbolic state space enumeration. Master's thesis, Helsinki University of Technology, Department of Computer Science, 1999.

15. KEMMERER, R., MEADOWS, C., AND MILLEN, J. Three systems for cryptographic protocol analysis. *Journal of Cryptology 7*, 2 (1994), 79–130.
16. LEWIS, H. R., AND PAPADIMITRIOU, C. H. *Elements of the Theory of Computation*. Prentice-Hall, Inc., 1981.
17. LOWE, G. An attack on the Needham-Schroeder public-key authentication protocol. *Information Processing Letters 56*, 3 (1995), 131–133.
18. ———. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems* (1996), Margaria and Steffen, Eds., no. 1055 in Lecture Notes in Computer Science, Springer-Verlag, pp. 147–166.
19. ———. Casper: A compiler for the analysis of security protocols. In *Proceedings of the Computer Security Foundations Workshop X* (1997), IEEE Computer Society Press.
20. ———. Towards a completeness result for model checking of security protocols (extended abstract). In *Proceedings of the Computer Security Foundations Workshop XI* (1998), IEEE Computer Society Press.
21. LOWE, G., AND ROSCOE, B. Using CSP to detect errors in the TMN protocol. *IEEE Transactions on Software Engineering 23*, 10 (1997), 659–669. Preliminary version published as Department of Mathematics and Computer Science Technical Report 1996/34, University of Leicester, 1996.
22. MEADOWS, C. A. Formal verification of cryptographic protocols: A survey. In *Advances in Cryptology — Asiacrypt '94* (1995), no. 917 in Lecture Notes in Computer Science, Springer-Verlag, pp. 133–150.
23. ———. Analyzing the Needham-Schroeder public key protocol: A comparison of two approaches. In *Proceedings of the Fourth European Symposium on Research in Computer Security — ESORICS 96* (1996), Springer-Verlag.
24. ———. The NRL protocol analyzer: An overview. *Journal of Logic Programming 26*, 2 (1996), 113–131.
25. MILLEN, J. K. The Interrogator model. In *Proceedings of the 1995 IEEE Symposium on Security and Privacy* (1995), IEEE Computer Society Press, pp. 251–260.
26. MITCHELL, J. C. Finite-state analysis of security protocols. In *Proceedings of the 10th International Conference on Computer Aided Verification — CAV '98* (1998), A. J. Hu and M. Y. Vardi, Eds., no. 1427 in Lecture Notes in Computer Science, Springer-Verlag, pp. 71–76.
27. MITCHELL, J. C., MITCHELL, M., AND STERN, U. Automated analysis of cryptographic protocols using Murφ. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy* (1997), IEEE Computer Society Press, pp. 141–151.
28. MITCHELL, J. C., SCHAMTIKOV, V., AND STERN, U. Finite-state analysis of SSL 3.0. In *Proceedings of the Sevent USENIX Security Symposium* (1998), pp. 201–215.
29. NEEDHAM, R. M., AND SCHRÖDER, M. D. Using encryption for authentication in large networks of computers. *Communications of the ACM 21*, 12 (Dec. 1978), 993–999.
30. PAULSON, L. C. Proving properties of security protocols by induction. In *Proceedings of the Computer Security Foundations Workshop X* (1997), IEEE Computer Society Press, pp. 70–83.
31. ROSCOE, A. W. Model-checking CSP. In *A Classical Mind, Essays in Honour of C. A. R. Hoare*. Prentice-Hall, 1994.
32. SCHNEIDER, S. Security properties and CSP. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy* (1996).
33. SHMATIKOV, V., AND STERN, U. Efficient finite-state analysis for large security protocols. In *Proceedings of the Computer Security Foundations Workshop XI* (1998), IEEE Computer Society Press.
34. STOLLER, S. D. Justifying finite resources for adversaries in automated analysis of authentication protocols. Technical Report 506, Indiana University, Computer Science, Mar. 1998. Revised February 1999.
35. ———. A reduction for automated verification of authentication protocols. Technical Report 520, Indiana University, Computer Science, Dec. 1998.
36. SYVERSON, P., AND VAN OORSCHOT, P. C. On unifying some cryptographic protocol logics. In *Proceedings of the 1994 IEEE Computer Society Symposium on Research in Security and Privacy* (1994), IEEE Computer Society Press, pp. 14–28.
37. WEDEL, G., AND KESSLER, V. Formal semantics for authentication logics. In *Proceedings of the Fourth European Symposium on Research in Computer Security — ESORICS 96* (1996), pp. 219–241.

A Needham-Schröder Public-Key Protocol Model

The following listing was used to model the Needham-Schröder Public-Key Protocol:

```
#
# Operators
#

declare-operators rsa-enc, rsa-dpub, rsa-dpriv, car, cons, cdr;

declare-PK-encryption         rsa-enc
  decrypted-by                rsa-enc
  with-public-key-deriver     rsa-dpub
  with-private-key-deriver    rsa-dpriv;

declare-tuple                 cons
  with-selectors              car, cdr;

#
# Atoms
#

declare-atoms alice, bob, igor, alice-keym, bob-keym, igor-keym, good, bad;

#
# Initial knowledge of the environment
#

declare-public alice, bob, igor;
declare-public rsa-dpub(alice-keym), rsa-dpub(bob-keym), rsa-dpub(igor-keym);
declare-public rsa-dpriv(igor-keym);

#
# Auxiliary procedures
#

procedure get-my-key [ID] local ALICE, BOB; {
  ALICE := alice; BOB := bob;
  if ID = ALICE { return rsa-dpriv(alice-keym); }
  if ID = BOB   { return rsa-dpriv(bob-keym); }
  halt;
}

procedure get-her-key [ID] local ALICE, BOB, IGOR; {
  ALICE := alice; BOB := bob; IGOR := igor;
  if ID = ALICE { return rsa-dpub(alice-keym); }
  if ID = BOB   { return rsa-dpub(bob-keym); }
  if ID = IGOR  { return rsa-dpub(igor-keym); }
  halt;
}

#
# Inititator
#

procedure ns-initiator [ID] {
  IDENTITY := ID;
```

```
  receive PEER;
  get-my-key [IDENTITY] -> MYKEY;
  get-her-key [PEER] -> HERKEY;
  generate MYNONCE;
  MESSAGEA := rsa-enc(HERKEY, cons(MYNONCE, IDENTITY));
  send MESSAGEA;

  receive RESPONSE;
  DECRYPTED := rsa-enc(MYKEY, RESPONSE);
  FIRSTNONCE := car(DECRYPTED);
  if FIRSTNONCE = MYNONCE {
    HERNONCE := cdr(DECRYPTED);
    MESSAGEB := rsa-enc(HERKEY, HERNONCE);
    send MESSAGEB;
    halt-check
      not[[exists Z
           [[not[Z.IDENTITY = ME.IDENTITY]]
            and Z.HERNAME = ME.IDENTITY
            and Z.STATUS = good
            and [not [ME.PEER = Z.IDENTITY]]]]];
  } else { halt; }
}

#
# Responder
#

procedure ns-responder [ID]
{
  IDENTITY := ID;
  get-my-key [IDENTITY] -> MYKEY;
  receive INITMSG;
  DECRYPTED := rsa-enc(MYKEY, INITMSG);
  HERNONCE := car(DECRYPTED);
  HERNAME := cdr(DECRYPTED);
  ALICENAME := alice;

  get-her-key [HERNAME] -> HERKEY;
  generate MYNONCE;
  MESSAGE := rsa-enc(HERKEY, cons(HERNONCE, MYNONCE));
  send MESSAGE;
  receive RESPONSE;
  DECRYPTEDB := rsa-enc(MYKEY, RESPONSE);
  if DECRYPTEDB = MYNONCE {
    STATUS := good; halt;
  } else {
    STATUS := bad; halt;
  }
}

#
# Role instances
#

principal 1 ns-initiator[alice];
principal 1 ns-responder[bob];
```