Efficient Finite-State Analysis for Large Security Protocols

Vitaly Shmatikov Ulrich Stern

Computer Science Department
Stanford University
Stanford, CA 94305-9045

{shmat,uli}@cs.stanford.edu

Abstract

We describe two state reduction techniques for finite-state models of security protocols. The techniques exploit certain protocol properties that we have identified as characteristic of security protocols. We prove the soundness of the techniques by demonstrating that any violation of protocol invariants is preserved in the reduced state graph. In addition, we describe an optimization method for evaluating parameterized rule conditions, which are common in our models of security protocols. All three techniques have been implemented in the Murφ verifier.

1 Introduction

Security protocols are becoming widely used and many new protocols are being proposed. Since security protocols are notoriously difficult to design, computer assistance in the design process is desirable. The existing verification methods are mainly based on either finite-state analysis, or computer-assisted proof. The two approaches are complementary. Unlike computer-assisted proof, finite-state analysis cannot guarantee correctness for a protocol of unbounded size (e.g., a protocol with a potentially unbounded number of participants). However, finite-state analysis requires much less human expertise and is fully automatic.

Finite-state analysis of security protocols begins with a high-level description of the honest participants of the protocol. This protocol model must be augmented with some specification of the possible actions of an intruder and a precise statement of the desired properties of the protocol. The finite-state analysis tool then exhaustively enumerates all reachable states of the model, checking for each state whether it satisfies the desired correctness criteria. The main problem in this analysis is the very large number of reachable states for most protocols.

In this paper, we describe two techniques that reduce the number of reachable states and hence allow the analysis of larger protocols. We prove the techniques sound, i.e., we show that each protocol error that would have been discovered in the original state graph will still be discovered in the reduced state graph. The techniques are based on certain protocol properties that we have identified as characteristic of security protocols. We have implemented both techniques in the Murφ verification system [3] and have evaluated them on the SSL [4] and Kerberos [5] protocols.

The first technique is to let the intruder always intercept messages sent by the honest participants (instead of making such interception optional). This technique has resulted in

a very large reduction in both the number of reachable states and execution time. While this technique has been used by several researchers [6, 1, e.g.], it has neither been proved sound, nor has its importance been demonstrated.

The second technique prevents the intruder from sending messages to honest participants in states where at least one of the honest participants is able to send a message. Intuitively, the technique makes the intruder more powerful since the intruder maximally increases its knowledge before forging and sending messages to honest participants; hence the analysis of the reduced state graph should not miss any attacks on the protocol. The technique typically saved a factor of two or more in the number of reachable states as well as execution time. It is interesting to note that that this technique is more powerful than partial-order techniques exploiting the independence of the honest participants.

In addition to the two state reduction techniques, we also describe a technique that reduces the execution time of Murφ, but not the number of reachable states. The technique is based on the following observations:

The intruder model employed in Murφ is highly nondeterministic and thus gives rise to a large number of state transition rules. In every reachable state, the enabling conditions of all rules are evaluated. Evaluation can be sped up by partitioning the rules into sets with identical enabling conditions and evaluating the condition only once for each set. This technique typically increased the overall speed of Murφ by a factor of four.

2 Overview of Murφ

Murφ [2] is a protocol or, more generally, finite-state machine verification tool. It has been successfully applied to several industrial protocols, especially in the domains of multiprocessor cache coherence protocols and multiprocessor memory models [3, 10, 11] and in the domain of security protocols [7, 8]. The purpose of finite-state analysis, commonly called "model checking," is to exhaustively search all execution sequences.

To verify a security protocol using Murφ, one has to model both the protocol and the intruder (or adversary) in the Murφ language and augment the resulting model with a specification of the desired properties. The Murφ system automatically checks, by explicit state enumeration, if every reachable state of the model satisfies the given specification. For the state enumeration, either breadth-first or depth-first search can be selected. Reached states are stored in a hash table to avoid redundant work when a state is revisited. The memory available for this hash table typically determines the largest tractable problem.

The intruder is generally modeled to control the network and allowed the following actions: (1) overhear every message, remember all parts of each message (in a knowledge database), and decrypt ciphertext when it has the key, (2) intercept (delete) messages, and (3) generate messages using any combination of initial knowledge about the system and parts of overheard messages. We also assume that the intruder can masquerade as an honest participant in the system, capable of initiating communication with a truly honest participant, for example. We will refer to this intruder model as "mechanical intruder model" because of its simplicity.

The Murφ language is a simple high-level language for describing nondeterministic finite-state machines. Many features of the language are familiar from conventional programming languages. The main features not found in a "typical" high-level language are described in the following paragraphs.

The *state* of the model consists of the values of all global variables. In a *startstate* statement, initial values are assigned to global variables. The transition from one state to another is performed by *rules*. Each rule has a Boolean condition and an action, which is a program segment that is executed atomically. The action may be executed if the condition

is true (i.e., the rule is enabled) and typically changes global variables, yielding a new state. Most Murφ models are nondeterministic since states typically allow execution of more than one rule. For example, in the model of a security protocol, the intruder usually has the nondeterministic choice of several messages to replay.

Murφ has no explicit notion of *processes*. Nevertheless a process can be implicitly modeled by a set of related rules. The *parallel composition* of two processes in Murφ is simply done by using the union of the rules of the two processes. Each process can take any number of steps (actions) between the steps of the other. The resulting computational model is that of *asynchronous, interleaving* concurrency. Parallel processes communicate via shared variables; there are no special language constructs for communication.

The Murφ language supports *scalable* models. In a scalable model, one is able to change the size of the model by simply changing constant declarations. When developing protocols, one typically starts with a small protocol configuration. Once this configuration is correct, one gradually increases the protocol size to the largest value that still allows verification to complete. In many cases, an error in the general (possibly infinite state) protocol will also show up in a scaled-down (finite state) version of the protocol. Murφ can only guarantee correctness of the scaled-down version of the protocol, but not correctness of the general protocol. For example, the numbers of clients and servers in a security protocol are typically scalable and defined by constants.

The desired properties of a protocol can be specified in Murφ by invariants, which are Boolean conditions that have to be true in every reachable state. If a state is reached in which some invariant is violated, Murφ prints an error trace – a sequence of states from the start state to the state exhibiting the problem.

3 Properties of security protocols

In this section, we identify several characteristic properties of security protocols that we will use to develop state reduction techniques. These properties characterize every security protocol we have encountered so far, including, e.g., Kerberos [5], SSL [4], and Needham-Schroeder [9]. The properties are quite simple, yet recognizing them is useful both for better understanding of the protocols and for making finite-state analysis as efficient as possible within the basic framework of the mechanical intruder model.

3.1 Protocol invariants are monotonic

The invariants used to specify correctness of security protocols are typically of the forms "Intruder does not know X" or "If honest participant A reaches state S_1, then honest participant B must be in state S_2." Assume that an invariant of this form is invalid for a given state. Then increasing the intruder's knowledge set (which is part of the state) will not make the violated invariant valid. Hence we will call the protocol invariant *monotonic* with respect to the intruder's *knowledge set*. All protocol invariants we have encountered to date have been monotonic.

The implication of this property for state reduction is that we can safely rearrange the reachable state graph, possibly eliminating some states, as long as we can guarantee that for every state in the old graph, there exists a state in the new graph in which the state of all honest protocol participants is the same while the intruder's knowledge set is the same or larger. Because protocol invariants are monotonic, such state reductions are *sound* in the sense that any invariant that would have been violated in the old state graph will still be violated in the new graph.

We also assume that protocol invariants are defined in terms of the intruder's knowledge set and the states of the honest protocol participants. The invariants should not depend on the state of the network. (Since the network is assumed to be controlled by the intruder, invariants that depend on the state of the network can be rewritten to depend on the intruder's knowledge set.)

3.2 Intruder controls the network

It is traditionally assumed in security protocol analysis that the intruder exercises full control over the network, including the option to intercept any message. Instead of giving the intruder the option to intercept messages, we will assume that the intruder always intercepts. We model interception in Murφ by having the intruder remove the message from the "network" and store it in its database. The intruder can then replay the message to the intended recipient, or forge a similar-looking message.

Intuitively, the assumption that every message gets intercepted will not weaken the intruder and should hence be sound. In Section 5.1 below, this assumption is used to cut the transitions leading to redundant states, differing only in the contents of the intruder's database. The only state left is the one in which the database contains all information observable from the exchange of protocol messages up to that moment.

3.3 Honest protocol participants are independent

Honest protocol participants are fully independent from each other. The only means of communication between the participants is by sending messages on the network, which is assumed to be fully controlled by the intruder. Sending a message to another participant is thus equivalent to simply handing it to the intruder, hoping that the latter will not be able to extract any useful information from it and will replay it intact to the intended recipient.

As a consequence, an honest protocol participant has no way of knowing for sure what the current state of other participants is, since all information about the rest of the world arrives to him through a network fully controlled by the intruder. Actions of each honest participant (i.e., sending and receiving of messages) thus depend only on its own local state and not on the global state that comprises the states of all participants plus that of the intruder. In our formal representation of security protocols as state graphs, we will rely on this property to make all transition rules for honest participants local (see Section 4.2 below). We do not consider protocols with out-of-band communication as they are beyond the scope of our research with Murφ.

4 Protocols as state graphs

In this section, we define a formalism for describing finite-state machines associated with security protocols.

4.1 States

The global state of the system is represented by a vector:

$$s = [s_1, \ldots, s_N, e]$$

where N is the number of honest protocol participants, s_i is the local state of the protocol participant i, and e is the state of the intruder.

Instead of modeling the global network, we model a separate 1-cell local network for each honest protocol participant. All messages intended for that participant are deposited in its local network as described in Section 4.2 below. The local state of an honest participant i is a pair

$$s_i = \langle v_i, m_i \rangle$$

where v_i is the vector of current values of i's local state variables, and m_i is the message currently in i's local network. It is possible that $m_i = \varepsilon$, representing the empty network.

The state of the intruder is simply the set of messages that the intruder has intercepted so far (assume that its initial knowledge is represented as an intercepted message also):

$$e = \{m_{e_1}, \ldots, m_{e_M}\}$$

The intruder's knowledge is obviously not limited to the intercepted messages. The intruder can split them into components, decrypt and encrypt fields, assemble new messages, etc. However, the full knowledge set can always be synthesized from the intercepted messages, since they are the only source of information available to the intruder. Therefore, our chosen representation for the intruder's state is sufficient to represent the intruder's knowledge. When necessary, we will refer to the set of messages that can be synthesized from the set of intercepted messages as synth(e). (Since operations like encryption and pairing can be applied infinitely many times in the synthesis, the intruder's full knowledge is generally infinite. In practice, one can extract finite limits on the numbers of times encryption and pairing have to be applied from the protocol definition, making the intruder's knowledge finite.)

4.2 Rules for honest participants

All transition rules between states have the following form in our formalism:

$$r_k^{(i|e)} = \text{if } c_k(s_i|e) \text{ then } s \rightarrow s'$$

where $c_k(s_i|e)$ is the condition of the rule (it depends on the local state s_i in case of an honest protocol participant, and the knowledge set e in case of the intruder), s is the original global state, s' is the global state obtained as the result of rule application.

We can assume without loss of generality that every transition rule for an honest protocol participant consists of reading a non-empty message off the local network, changing the local variables, and sending a non-empty message to another participant. If necessary, the protocol can be rewritten so as to avoid "hidden" transitions that change the state of a participant without visible activity on the network. The initial transition for each participant can be triggered by a special message deposited into its local network in the start state of the system, and the last transition can be rewritten so that it deposits another special message on the network. This ensures that every transition reads and writes into the network. Also, the 1-cell capacity restriction on the local network is not essential, since we will eventually assume that every message is intercepted by the intruder immediately after it has been sent.

The transition rules for an honest protocol participant i are represented as follows:

$$r_k^{(i)} = \text{if } c_k(v_i, m_i) \text{ then } [\ldots\langle v_i, m_i\rangle \ldots \langle v_j, \varepsilon\rangle \ldots] \rightarrow [\ldots\langle v_i', \varepsilon\rangle \ldots \langle v_j, m_j\rangle \ldots]$$

Informally, if condition c_k evaluates to `true` given i's local state $s_i = \langle v_i, m_i \rangle$, then i reads message m_i off its local network, executes some code changing its local state variables

from v_i to v_i', and sends message m_j to participant j by depositing it in j's local network. Note that the rule is local — its condition depends only on i's local state. We assume that honest participants are deterministic. In any state, there is no more than one rule enabled for each participant. However, it is possible that rules for several participants are enabled in the same state, resulting in nondeterminism.

4.3 Rules for the intruder

The transition rules for the intruder are global. The first set of rules describes the intruder intercepting a message intended for an honest participant:

$$r_{-_i}^{(e)} = \text{if origin}(m) \neq e \text{ then } [\ldots\langle v_i, m\rangle\ldots e] \ \rightarrow\ [\ldots\langle v_i, \varepsilon\rangle\ldots e \cup \{m\}]$$

The intruder first checks the origin of the message on i's local network, since we do not want the intruder to remove its own messages. If the message was generated by an honest participant, it is removed from the network and added to the intruder's database. Note that the local variables of the honest participant are not affected by this action.

The second set of intruder rules describes the intruder generating a message and sending it to an honest participant whose local network is empty:

$$r_{+_i}^{(e)} = \text{if } \texttt{true} \text{ then } [\ldots\langle v_i, \varepsilon\rangle\ldots e] \ \rightarrow\ [\ldots\langle v_i, m\rangle\ldots e] \qquad m \in \text{synth}(e)$$

The intruder creates a new message (either by replaying an intercepted message $m \in e$, or by synthesizing m from the messages stored in e) and sends it to participant i by depositing it in i's local network. Clearly, the intruder rules are nondeterministic, as there may be several intruder rules enabled in the same state.

There are no other transition rules in the system.

4.4 State graph

The finite-state machine associated with the protocol is a directed graph $\{S, T, s_0, Q\}$. The vertices S are all possible states of the protocol. The directed edges T are pairs of states labeled by rules such that $r_k^{(i|e)} : \langle s, s'\rangle \in T$ *iff* the transition rule $r_k^{(i|e)}$ is enabled in state s (i.e., its condition evaluates to `true`) and transforms s into state s'. The finite-state machine is nondeterministic. If several rules are enabled in state s, there will be several edges leaving the corresponding graph vertex. We will refer to the subgraph reachable from vertex s as $R(s)$.

$s_0 \in S$ is the start state of the protocol.

Q is the set of protocol invariants. An invariant is a function from states to boolean values $q : S \rightarrow \texttt{true}|\texttt{false}$ such that $q(s) = \texttt{false}$ if the invariant q is violated in s, otherwise $q(s) = \texttt{true}$. Since protocol invariants do not depend on the state of the network, the value of q in any state does not depend on the m_i values representing the current contents of the participants' local networks.

We will say that state $s = [s_1 \ldots s_N, e]$ is *subsumed* by another state $s' = [s_1' \ldots s_N', e']$ *iff* $\forall i\ s_i = s_i'$ and $e \subseteq e'$. Informally, s' subsumes s if the states of all honest participants are the same in s' as in s while the intruder's knowledge set is the same or larger. In the rest of the paper, subsumption will be denoted as $s \preceq s'$.

Note that if $s \preceq s'$, then all rules enabled in s are enabled in s', too. The reverse is not true since the intruder's knowledge set is larger in s' and some rules may be enabled in s' but not in s. Therefore, $R(s)$ (set of states reachable from s) is isomorphic to a subgraph of $R(s')$. Thus, for any t reachable from s, there exists a t' reachable from s' such that $t \preceq t'$. We will refer to this fact as *inheritance of subsumption*.

Protocol invariants are monotonic with respect to the intruder's knowledge set (increasing intruder's knowledge does not repair any violated invariants). Therefore, if $s \preceq s'$, then $q(s) = \texttt{false}$ implies that $q(s') = \texttt{false}$. We will refer to this fact as *monotonicity of invariance.*

4.5 Soundness of state reduction

A finite-state analyzer such as Murφ verifies the protocol by traversing the state graph starting from state s_0. For every state s it reaches, Murφ verifies that all invariants are valid, i.e., $\forall q\ q(s) = \texttt{true}$. Violation of any invariant signals an error in the protocol.

A state reduction technique transforms the original state graph $\{S, T, s_0, Q\}$ into a new graph $\{S', T', s_0, Q\}$. We claim that the technique is *sound* if all errors that would have been discovered in the original graph will still be discovered in the new graph. More formally, if the old graph contains a reachable state in which one of the invariants was violated, then the new graph should contain a reachable state in which the *same* invariant is violated.

$$\exists s \in R(s_0)\ \ \exists q \in Q\ \ \text{s.t.}\ \ q(s) = \texttt{false} \qquad \text{implies} \qquad \exists s' \in R'(s_0)\ \ \text{s.t.}\ \ q(s') = \texttt{false}$$

In the soundness proofs for the state reduction techniques below, we will demonstrate that the state graph contains two vertices s and s' such that $s \preceq s'$. By monotonicity of invariance, all invariants that are violated in s are also violated in s'. Moreover, by inheritance of subsumption, for any t reachable from s, there exists a t' reachable from s' such that all invariants violated in t are also violated in t'.

Suppose that we find a vertex s^* in the state graph such that there are edges leading from s^* to both s and s'. We cut the edge from s^* to s. The subgraph rooted in s may become unreachable, reducing the number of states to be explored. However, we do not "lose" any protocol errors by eliminating these states. Every eliminated state in which an invariant is violated has a counterpart reachable from s' in which the same invariant is violated.

5 State reduction techniques

In this section, we describe the state reduction techniques and prove them sound.

5.1 Intruder always intercepts

Consider a state in which one of the rules for honest participants is enabled:

$$s_1 = [\ldots \langle v_i, m_i \rangle \ldots \langle v_j, \varepsilon \rangle \ldots e]$$

Suppose that state s_1 is such that condition $c_k(v_i, m_i)$ evaluates to $\texttt{true}$. Then rule $r_k^{(i)}$ (participant i sends message m_j to participant j) is enabled in s_1. Suppose that message m_i is not known to the intruder, i.e., $m_i \notin e$. We intend to reduce the number of states to be explored by having the intruder always intercept message m_i.

Fig. 1 shows the subgraph rooted in s_1, where

$$\begin{aligned} s_2 &= [\ldots \langle v_i', \varepsilon \rangle \ldots \langle v_j, m_j \rangle \ldots e] \\ s_3 &= [\ldots \langle v_i, \varepsilon \rangle \ldots \langle v_j, \varepsilon \rangle \ldots e \cup \{m_i\}] \\ s_4 &= [\ldots \langle v_i, m_i \rangle \ldots \langle v_j, \varepsilon \rangle \ldots e \cup \{m_i\}] \\ s_5 &= [\ldots \langle v_i', \varepsilon \rangle \ldots \langle v_j, m_j \rangle \ldots e \cup \{m_i\}] \end{aligned}$$

Each transition in Fig. 1 is labeled with the corresponding rule: rule $r_k^{(i)}$ is enabled both in s_1 and s_4; rule $r_{-i}^{(e)}$ (intruder removes a message from i's local network and stores it in the

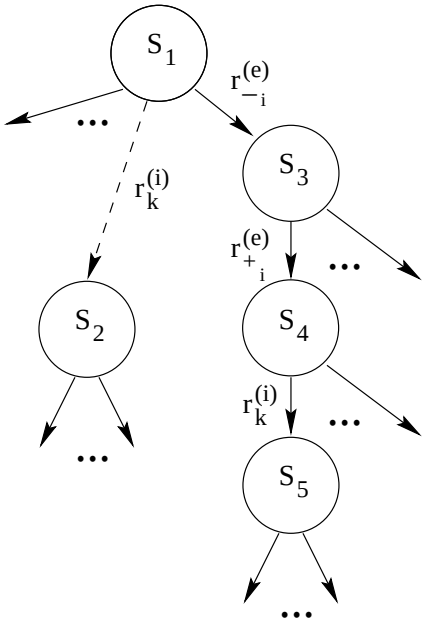

Figure 1: Intruder always intercepts

database) is enabled in s_1; rule $r^{(e)}_{+_i}$ (intruder deposits a message from its database into i's local network) is enabled in s_3. There may be additional rules enabled in states $s_1, \ldots, s_5$, but we will limit our attention to the subgraph shown in the figure.

We now observe that $s_2 \preceq s_5$ (the states of all honest participants are the same but the intruder's knowledge set is larger in s_5). Therefore, we can remove the graph edge leading from s_1 to s_2. Any invariant violation that can be discovered by analyzing $R(s_2)$ will be discovered by analyzing $R(s_5)$.

This state reduction technique effectively eliminates direct communication between participants. Every message sent on the network is intercepted by the intruder and added to the intruder's database. There is no need to consider states in which the database is "incomplete" (s_2 in the example above) since they are subsumed by the states in which the database is as complete as possible (s_5 in the example above), containing all messages exchanged on the network so far.

We can simplify the state transition rules slightly by assuming that rule $r^{(i)}_k$ deposits the generated message m_j directly into the intruder's database (recall that the original rule deposited the message into j's local network).

$$\hat{r}^{(i)}_k = \text{if } c_k(v_i, m_i) \text{ then } [\ldots\langle v_i, m_i\rangle\ldots\langle v_j, \varepsilon\rangle\ldots e] \ \rightarrow\ [\ldots\langle v_i', \varepsilon\rangle\ldots\langle v_j, \varepsilon\rangle\ldots e \cup \{m_j\}]$$

This simplification also eliminates the need for $r^{(e)}_{-_i}$ rules. Another consequence is that $r^{(e)}_{+_i}$ are the only rules that can deposit a message into an honest participant's local network.

5.2 Intruder does not send if honest participant can send

Fig. 2 shows a fragment of the state graph in which

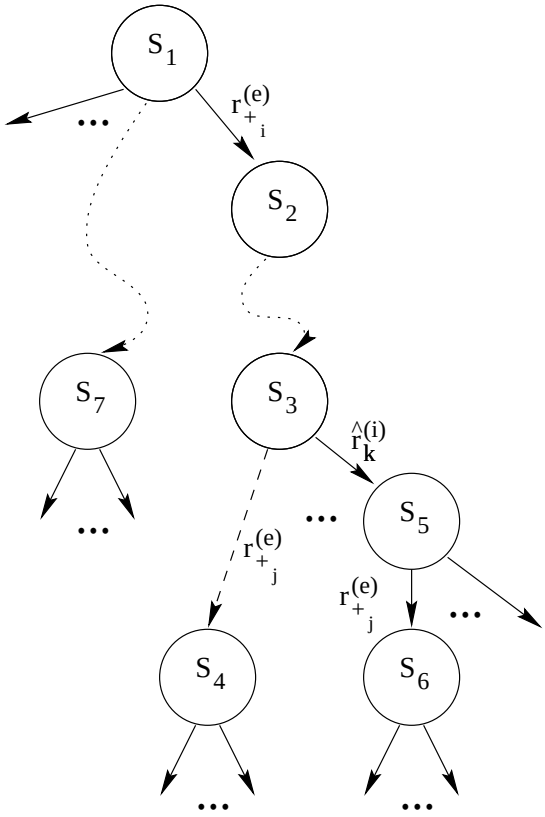

Figure 2: Intruder does not send if honest participant can send

$$
\begin{array}{lcl}
s_1 & = & [\ldots\langle v_i, \varepsilon\rangle \ldots \langle \ldots, \ldots\rangle \ldots e] \\
s_2 & = & [\ldots\langle v_i, m_i\rangle \ldots \langle \ldots, \ldots\rangle \ldots e] \\
s_3 & = & [\ldots\langle v_i, m_i\rangle \ldots \langle v_j, \varepsilon\rangle \ldots e] \\
s_4 & = & [\ldots\langle v_i, m_i\rangle \ldots \langle v_j, m_j\rangle \ldots e] \\
s_5 & = & [\ldots\langle v'_i, \varepsilon\rangle \ldots \langle v_j, \varepsilon\rangle \ldots e \cup \{m'_j\}] \\
s_6 & = & [\ldots\langle v'_i, \varepsilon\rangle \ldots \langle v_j, m_j\rangle \ldots e \cup \{m'_j\}] \\
s_7 & = & [\ldots\langle v_i, \varepsilon\rangle \ldots \langle v_j, m_j\rangle \ldots e] \\
 & & \text{where } m_i, m_j \in \text{synth}(e),\ m'_j \notin e
\end{array}
$$

In state s_3, at least two rules are enabled: $r^{(e)}_{+_j}$ corresponds to the intruder depositing m_j into j's network, and $\hat{r}^{(i)}_k$ corresponds to the intruder waiting for participant i to send its message first. We intend to demonstrate that the number of states to be explored can be reduced by considering only the latter case. Formally, the edge from s_3 to s_4 can be cut from the state graph since any violation of protocol invariants that can be discovered in $R(s_4)$ will be discovered either in $R(s_6)$, or in $R(s_7)$,

Suppose that there exists a state $t_E \in R(s_4)$ such that $q(t_E) = \texttt{false}$ for some invariant $q \in Q$. Consider the sequence of state transitions that leads from s_4 to t_E. Each transition is labeled by the corresponding rule:

$$s_4 \xrightarrow{r_3} t_0 \xrightarrow{r_1} t_1 \rightarrow \ldots \xrightarrow{r_E} t_E$$

We will consider two cases.

Case 1. Suppose $\exists L \in 0..E\ r_L = \hat{r}^{(i)}_k$, and none of the rules r_l for $l \in 0..L-1$ involve participant i. In other words, rule $\hat{r}^{(i)}_k$ is executed at some point between s_4 and t_E.

Consider that state s_6 is identical to s_4 except for the state of participant i, which does not matter for rules r_l, and the intruder's knowledge set, which is strictly larger in s_6. Therefore, the rule sequence r_l is enabled in s_6, leading to a state $\tilde{t}_{L-1}$. To complete the proof for this case, observe that t_L is equivalent to $\tilde{t}_{L-1}$.

Case 2. Suppose that $\forall l \in 0..E\ r_l \neq \hat{r}^{(i)}_k$, i.e., none of rules leading to the erroneous state involve participant i reading message m_i off the network and sending message m'_j to participant j. The only rule enabled for i is $\hat{r}^{(i)}_k$, and it cannot become disabled as long as the local state of i does not change. Therefore, none of the rules r_l involve participant i at all, and s_i is the same in t_E as in s_4. To prove soundness for this case, we will first demonstrate that if the state graph includes state s_3, it must also include state s_7, and then we will show that if an invariant violation can be discovered in $R(s_4)$, it can also be discovered in $R(s_7)$.

To prove that if the state graph contains s_3, it must also contain s_7, consider the sequence of rules that leads from the start state s_0 to s_3. Since participant i's local network contains m_i in it, the last rule in the sequence that involves i must be $r^{(e)}_{+_i}$, since only the intruder can deposit a message into an honest participant's local network. In Fig. 2, s_1 represents the state to which rule $r^{(e)}_{+_i}$ was applied, and s_2 represents the resulting state. Note that none of the rules leading from s_2 to s_3 involve i, since $r^{(e)}_{+_i}$ was the *last* such rule in the sequence. Therefore, the same rules are enabled in state s_1 that differs from s_2 only in the contents of i's local network. Applying the rules, we obtain the state $\tilde{s}$ that differs from s_3 only in the contents of i's local network, namely, i's network is empty in $\tilde{s}$ but contains m_i in s_3. Therefore, $\tilde{s} = s_7$.

To complete the soundness proof, observe that state s_7 is identical to s_4 except for the state of participant i. Since none of the rules r_l leading to t_E involve participant i, the same sequence of rules is enabled in s_7, leading to a state $\tilde{t}_E$ such that the only difference between $\tilde{t}_E$ and t_E is the state of participant i. More precisely, i's local network is empty in $\tilde{t}_E$ but contains m_i in t_E. This implies that $q(\tilde{t}_E) = \texttt{false}$, since invariants do not depend on the state of the network. The proof is complete.

The described state reduction technique makes sure that the intruder never sends a message if there is an honest participant who is ready to send a message, too (i.e., one of the rules for honest participants is enabled). The intuitive reason for this is that the intruder's knowledge set should be as complete as possible before the intruder synthesizes a message. By waiting until the honest participant sends its message and intercepting it, the intruder potentially increases its knowledge set.

To ensure that the intruder sends messages only when no one else can, we can modify the condition for rule $r_{+i}^{(e)}$ so that it is enabled only in the states of the following form:

$$s = [\langle v_1, \varepsilon \rangle, \ldots, \langle v_N, \varepsilon \rangle, e]$$

An alternative implementation of this state reduction technique is to assign a lower priority to the intruder rules than to the rules for honest participants (see Section 6.1 below).

Remark. A related state reduction technique exploits the independence of honest protocol participants to fix a partial order and eliminate multiple interleavings of message sends. Since all participants send their messages to the intruder, the order in which the messages are deposited into the intruder's database does not matter.

However, there is no need to impose a partial order on the honest participants if the technique described in this section is implemented. To see why this is the case, suppose we start with an empty network. After the intruder sends a message to the first participant, the rules for the intruder become disabled until the first participant replies by depositing its message into the intruder's database. Only then can the intruder send a message to another participant. Hence there are no states in which more than one participant is ready to send a message.

6 Implementation issues

In this section, we discuss how state reduction techniques can be implemented by assigning priorities to transition rules. We also describe another Murφ optimization that does not rely on state reduction.

6.1 Rule priorities

To support the state reduction techniques described above, we extended Murφ language with *rule priorities*. In the extended language, a model implementor can assign an integer priority to every rule in the system. If several rules are enabled in a particular state, Murφ will only explore the subgraphs corresponding to rules with the highest priority. User-specified rule priorities help Murφ recognize which of the rules are associated with honest participants and which are intruder rules.

The technique from Section 5.1 (intruder always intercepts) can be implemented by assigning the highest priority to the "Intruder intercepts" rule. Then, whenever a new

message appears on the network, it is immediately intercepted by the intruder and stored in its database.

The technique from Section 5.2 (intruder does not send if an honest participant can) can be implemented by assigning the lowest priority to the rules for intruder sending a message. The only other rules in the system are those associated with honest participants, and each of those rules sends a message to the network. Therefore, as long as there exists an honest participant who is ready to send a message, the intruder will not send messages but will intercept those from honest participants, increasing its knowledge set.

6.2 Parameterized rule conditions

After an honest participant has sent its message, the system reaches a state in which the only enabled rules are those representing the intruder sending messages to honest participants. Typically, the structure of the messages to be sent is known from the protocol specification. However, the mechanical intruder employed in Murφ cannot determine on its own what values have to be assigned to message fields so that the resulting message is accepted by the recipient and potentially leads to a successful attack. Therefore, the intruder will try all possible combinations of field values. For example, the following rule represents the intruder forging a *Finished* message in SSL 3.0 (*Finished* messages contain a record of all information previously sent in the protocol):

```
-- Intruder generates a ServerFinished message with
-- a forged handshake log

ruleset d: ClientId do
  choose  n1: int.nonces do
  choose  n2: int.nonces do
  choose  secretKey: int.secretKeys do
  ruleset sender: ServerId do
    rule "Intruder generates ServerFinished (forged log)"

      cli[d].state = M_SERVER_FINISHED

    ==>

      -- Forge a message with the above parameters
      ...
end
```

In the above rule, the intruder nondeterministically chooses (using the `ruleset` and `choose` constructs) a recipient `d` from among the known SSL clients, two nonces `n1` and `n2` from its database of nonces, a `secretKey` from its database of keys, and the ostensible `sender` of the message from among the SSL servers, and finally forges and sends the message if the recipient is in a state in which it is ready to receive the message. (The intruder is assumed to know the states of all honest protocol participants since the states can be inferred from the protocol definition and the observed message exchange.)

The above Murφ code defines a rule for each possible assignment to the parameters `d`, `n1`, `n2`, `secretKey`, and `sender`. The number of defined rules is thus equal to the product of the cardinalities of the sets from which the parameter values are drawn. Clearly, if the intruder

databases are sufficiently large and there are many choices for each of the parameters, the number of defined rules is large.

In each state reached during the state space search, Murφ checks for each rule whether it is enabled or not. Many rules, however, share the same condition. In the example above, all rules generated for the same value of `d` have the same condition, regardless of the values of the other parameters. Thus, it is sufficient to evaluate the condition just once for each possible assignment to `d`.

To exploit this idea, we modified the Murφ compiler so that it separates the parameters for each `rule` statement into two sets. The *Dep* set contains all parameters that are mentioned in the rule condition (`d` in the example above). The *Indep* set contains the parameters *not* mentioned in the rule condition (`n1`, `n2`, `secretKey`, and `sender` in the example above).

As before, a separate rule is defined for every possible assignment to the parameters from $Dep \cup Indep$. The set of defined rules is partitioned into subsets so that each subset corresponds one-to-one to a particular assignment to the parameters from *Dep*. Therefore, each subset contains the rules corresponding to all possible assignments to the parameters from *Indep* given a particular assignment to the parameters from *Dep*.

Instead of evaluating every rule condition in each state, the Murφ verifier was altered to only evaluate the condition for one (arbitrarily chosen) rule from each rule subset. The result of evaluation is the same for all rules in the subset since they differ only in the values of the parameters from *Indep*, and the conditions do not depend on those parameters. Therefore, if the condition evaluates to `true`, all rules in the subset are executed. If the condition evaluates to `false`, the verifier immediately moves on to the next subset corresponding to the next assignment to the parameters from *Dep*.

In our experience with security protocols, *Dep* is usually much smaller than *Indep*. The rules with the largest number of parameters are those representing the intruder's sending a message to an honest participant. The parameters of such rules contain values (chosen from the intruder's database) for the message fields of the forged message, while the rule conditions depend only on the current state of the recipient and not on the contents of the intruder's database. In fact, the intruder's knowledge should not have any influence on whether an honest participant is ready to receive a particular message or not. Therefore, not evaluating rule conditions for every possible assignment to the rule parameters has proved very profitable in our Murφ analyses.

6.3 Preliminary results

We used the state reduction techniques described in this paper to reduce the size of the finite-state model for SSL 3.0, which is the largest security protocol analyzed with Murφ to date. The first state reduction technique (intruder always intercepts) reduced the number of states by a factor of 20. Although spectacular, a reduction of this magnitude was to be expected. In the original model, every message could be intercepted by the intruder or delivered directly from sender to recipient. Intuitively, a sequence of N messages resulted in 2^N states, since each of the N messages could be intercepted by the intruder or not.

We profiled the resulting model and determined that most of the execution time was spent evaluating rule conditions due to the very large number of generated rules. Optimizing rule condition evaluation as described in Section 6.2 reduced execution time by a factor of 3.7 with the same number of states.

Finally, we implemented the second state reduction technique (intruder does not send if an honest participant can), which resulted in further 43% reduction in the number of states and a 40% reduction in the execution time.

We also evaluated our techniques on the Kerberos protocol, starting with the efficient

condition evaluation, and then adding the two state reduction techniques. Table 1 shows the numbers of reachable states and execution times dependent on the numbers of clients and servers in the protocol. As in the case of SSL, the biggest savings result from the first state reduction technique. Note that the savings resulting from the second state reduction technique increase with increasing numbers of clients and servers, as one would expect.

Table 1: Numbers of reachable states and execution times dependent on the model parameters in the Kerberos protocol

clients	servers		previous scheme	efficient conditions	always intercept	intruder send low priority
1	2	states	14317	14317	232	175
		time	191.8s	68.9s	2.0s	1.6s
2	2	states			541	193
		time			4.4s	2.2s
3	3	states			856	195
		time			12.3s	3.6s

7 Conclusion and future research

We described two state reduction techniques that exploit characteristic properties of security protocols. These techniques reduce both the number of reachable states and execution time of finite-state analysis. In addition, we described a method for minimizing the time required to evaluate parameterized rule conditions, further reducing total execution time.

The techniques described in this paper have proved very useful for analyzing large security protocols in Murφ. While the first state reduction technique (intruder always intercepts) has been employed in other finite-state analysis tools, the second state reduction technique (intruder does not send if an honest participant can) is novel and is expected to prove useful beyond the Murφ community.

Future research includes extending the Murφ verifier so that it can automatically recognize subsumption relations between states and remove subsumed states from the state queue. This technique is independent from the second state reduction technique described in this paper. We expect that combining the two techniques will result in larger state reductions than those achieved by either of the two techniques on its own. In addition, we plan to exploit the fact that, when the second state reduction technique is implemented, every message send from the intruder is immediately followed by a corresponding reply from the honest participant. This allows, for example, to combine send-reply pairs into a single intruder rule, which should result in a significant reduction in the number of reachable states. We also intend to explore other properties of security protocols and derive new state reduction techniques from them.

References

[1] D. Bolignano. Towards a mechanization of cryptographic protocol verification. In *Computer Aided Verification. 9th International Conference*, pages 131–42, 1997.

[2] D. L. Dill. The Murφ verification system. In *Computer Aided Verification. 8th International Conference*, pages 390–3, 1996.

[3] D. L. Dill, S. Park, and A. G. Nowatzyk. Formal specification of abstract memory models. In *Symposium on Research on Integrated Systems*, pages 38–52, 1993.

[4] A. O. Freier, P. Karlton, and P. C. Kocher. The SSL protocol version 3.0. `draft-ietf-tls-ssl-version3-00.txt`, November 18, 1996.

[5] J.T. Kohl and B.C. Neuman. The Kerberos network authentication service (version 5). Internet Request For Comments RFC-1510, September 1993.

[6] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using CSP and FDR. In *2nd International Workshop on Tools and Algorithms for the Construction and Analysis of Systems*. Springer-Verlag, 1996.

[7] J. C. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using Murφ. In *IEEE Symposium on Security and Privacy*, pages 141–51, 1997.

[8] J. C. Mitchell, V. Shmatikov, and U. Stern. Finite-state analysis of SSL 3.0. In *7th USENIX Security Symposium*, pages 201–15, 1998.

[9] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–9, 1978.

[10] U. Stern and D. L. Dill. Automatic verification of the SCI cache coherence protocol. In *Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, pages 21–34, 1995.

[11] L. Yang, D. Gao, J. Mostoufi, R. Joshi, and P. Loewenstein. System design methodology of UltraSPARC™-I. In *32nd Design Automation Conference*, pages 7–12, 1995.