Model Checking for Security Protocols

Will Marrero Edmund Clarke Somesh Jha

May 1997

CMU-CS-97-139

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

As more resources are added to computer networks, and as more vendors look to the World Wide Web as a viable marketplace, the importance of being able to restrict access and to insure some kind of acceptable behavior even in the presence of malicious intruders becomes paramount. People have looked to cryptography to help solve many of these problems. However, cryptography itself is only a tool. The security of a system depends not only on the cryptosystem being used, but also on *how* it is used. Typically, researchers have proposed the use of security protocols to provide these security guarantees. These protocols consist of a sequence of messages, many with encrypted parts. In this paper, we develop a way of verifying these protocols using *model checking*. Model checking has proven to be a very useful technique for verifying hardware designs. By modelling circuits as finite-state machines, and examining all possible execution traces, model checking has found a number of errors in real world designs. Like hardware designs, security protocols are very subtle, and can also have bugs which are difficult to find. By examining all possible execution traces of a security protocol in the presence of a malicious intruder with well defined capabilities, we can determine if a protocol does indeed enforce its security guarantees. If not, we can provide a sample trace of an attack on the protocol.

This research was sponsored in part by the Avionics Laboratory, Wright Research and Development Center, Aeronautical Systems Division (AFSC), U.S. Air Force, Wright-Patterson AFB, Ohio 45433-6543 under Contract F33615-90-C-1465, ARPA Order No. 7597, and in part by the National Science Foundation under grant no. CCR-8722633 .

Keywords: computer security, cryptographic protocols, formal verification, model checking, partial order.

1 Introduction

Security for early computers was provided by their physical isolation. Unauthorized access to these machines was prevented by restricting physical access. The importance of sharing computing resources led to systems where users had to authenticate themselves, usually by providing a name/password pair. This was sufficient if the user needed to be physically at the console or was connected to the machine across a secure link. However, the efficiency to be gained by sharing data and computing resources has led to computer networks, in which the communication channels cannot always be trusted. In this case, authentication information such as the name/password pairs could be intercepted and even replayed to gain unauthorized access. When such networks were local to a certain user community and isolated from the rest of the world, many were willing take this risk and to place their trust in the community. However, in order to be able to share information with those outside the community, this isolation would have to be removed. The benefits to be had by such sharing have been enormous, and the gains are demonstrated by the growth of such entities as the Internet and the World Wide Web. Now, very few, if any guarantees can be made about the communication links. Numerous protocols that take advantage of cryptography have been proposed that claim to solve many of the security issues. The correctness of these protocols is paramount, especially when we consider the size of the networks involved and the desire of users to place confidential information and to allow for monetary transactions to take place across these networks.

Typically, these protocols can be thought of as a set of principals which send messages to each other. The hope is that by requiring agents to produce a sequence of messages, the security goals of the protocol can be achieved. For example, if a principal A receives a message encrypted with a key known only by principals A and B, then principal A should be able to conclude that the message originated from principal B. However, it would be incorrect to conclude that principal A is talking to principal B. An adversary could be replaying a message overheard during a pervious conversation between A and B. So, depending on the security goal of this simple example protocol, the protocol may or may not be secure. Because the reasoning behind the correctness of these protocols can be subtle, a number of researchers have turned to formal methods to prove protocols correct.

In order to concentrate on the security of the protocol itself as opposed to the the security of the cryptosystem used, the vast majority of research in this area has made the following "perfect encryption" assumptions.

- The decryption key must be known in order to extract the plaintext from the cyphertext.
- There is enough redundancy in the cryptosystem that a cyphertext can only be generated using encryption with the appropriate key. This also implies that there are no encryption collisions. If two cyphertexts are equal, they must have been generated from the same plaintext using the same key.

While the assumptions are obviously not true, they are, in practice, reasonable. They are important because they allow us to abstract away the cryptosystem and analyze the protocol itself. In particular, if there is an attack on this abstracted protocol, then the same attack exists when a real cryptosystem is used.

2 Related Work

Because these protocols tended to be short and not terribly complicated, informal arguments were used to prove their correctness. However, when running in parallel, the behavior of these protocols is more difficult to analyze. Asynchronous composition is already difficult to reason about, and adding issues of who knows what and when makes reasoning about security protocols extremely difficult. One recent approach taken by Bellare and Rogaway and by Shoup and Rubin, is to try to provide a rigorous mathematical proof of the correctness of a protocol [3, 21]. They use properties of pseudo-random functions and mathematical arguments to prove that an adversary does not have a statistical advantage when trying to discover a key in a session key distribution protocol.

One of the earliest successful attempts at formally reasoning about security protocols involved developing a new logic in which one could express and deduce security properties. The earliest such logic is commonly referred to as the BAN logic and is due to Burrows, Abadi, and Needham [6]. Their syntax provided constructs for expressing intuitive properties like “A said X,” “A believes X,” “K is a good key,” and “S is an authority on X.” They also provide a set of proof rules which can then be used to try to deduce security properties like “A and B believe K is a good key” from a list of explicit assumptions made about the protocol. This formalism was successful in uncovering implicit assumptions that had been made and weaknesses in a number of protocols. However, this logic has been criticized for the “protocol idealization” step required when using this formalism. Protocols in the literature are typically given as a sequence of messages. Use of the BAN logic requires that the user transform each message in the protocol into formulas *about* that message, so that the inferences can be made within the logic. For example, if the server sends a message containing the key K_{ab}, then that step might need to be converted into a step where the server sends a message containing $A \overset{K_{ab}}{\leftrightarrow} B$, meaning that the key K_{ab} is a good key for communication between A and B. An attempt to give this logic a rigorous semantics was made by Abadi and Tuttle [2] and other attempts to improve or expand the logic can be found in [22]. The BAN logic remains popular because of its simplicity and high level of abstraction.

Recent work in the use of modal logics for verifying security protocols includes the development of a logic that can express accountability [13]. Kailar convincingly argues that in applications such as electronic commerce, it is accountability and not belief that is important. Like their counterparts in the paper world, one would like people to be held accountable for their electronic transactions. This means that it is not enough for the individual participants to believe that a transaction is taking place. They must be able to prove to a third party that a transaction is taking place. Kailar provides a syntax which allows such properties to be expressed and a set of proof rules for verifying them. Similar to the BAN logic, Kailar's accountability logic is at a very high level of abstraction. Still, Kailar is able to use it to analyze four protocols and to find a lack of accountability in a variant of one of CMU's Internet Billing Server Protocols.

An orthogonal line of research revolves around trying to automate the process of verification when using these logics. Craigen and Saaltink attempt this by embedding the BAN logic in EVES [7]. The automation resulting from this experiment was not satisfactory. By building a forward-chaining mechanism and changing some of the rules, they were able to build a system that would try to develop the entire theory of a set of axioms (find the closure of a set of formulas under the derivation rules). Kindred and Wing went further by proposing a theory-checker *generator* [14]. They provide a formal and well defined framework with assurances about correctness and termination. In addition, their system generates theory checkers for a variety of logics including BAN, AUTLOG, and Kailar's accountability logic.

The third technique can be placed in the general category of model checking. The common approach here is to model the protocol by defining a set of states and a set of transitions that takes into account an intruder, the messages communicated back and forth, and the information known by each of the principals. This state space can then be traversed to check if some particular state can be reached or if some state trace can be generated. The first attempt at such a formalism is due to Dolev and Yao [8]. They develop an algorithm for determining whether or not a protocol is secure in their model. However, their model is extremely limited. They only consider secrecy issues, and they model only encryption, decryption, and adding, checking, or deleting a principal name.

Meadows used an extension of the Dolev-Yao model in her PROLOG based model checker [17]. In her system, the user models a protocol as a set of rules that describe how an intruder generates knowledge. These rules model both how the intruder can generate knowledge on its own by applying encryption and decryption, and how the intruder can generate new knowledge by receiving responses to messages it sends to the principals participating in the protocol. In addition, the user specifies rewrite rules that indicate how words are reduced. Typically, there are three rules used to capture the notion of equality and the fact that encryption and decryption are inverse functions. These rules are:

encrypt(X,decrypt (X,Y)) $\rightarrow$ Y
decrypt(X,encrypt (X,Y)) $\rightarrow$ Y
id_check(X,X) $\rightarrow$ yes

To perform the verification, the user supplies a description of an insecure state. The model checker then searches backwards in an attempt to find an initial state. This is accomplished naturally in PROLOG by attempting to unify the current state against the right hand side of a rule and thus deducing from the left hand side what the state description for the previous state must be. If the initial state is found, then the system is insecure, otherwise an attempt is made to prove that the insecure state is unreachable by showing that any state that leads to this particular state is also unreachable. This kind of search often leads to an infinite trace where in order for the intruder to learn word A, it must learn word B, and in order to learn word B, it must learn word C, and so on. For this reason a facility for formal languages is included which allows the user to prove that no word in a set of words (or language) can be generated by the intruder. The technique involves the following steps:

- Show that the word in question is in the language.
- Show that knowledge of any word in the language requires previous knowledge of another word in the language.
- Show that the initial state does not contain any word in the language.

This initial model checker was still too limited. In particular, it did not allow the modeling of freshly generated nonces or session keys. The model checker evolved into the NRL Protocol Analyzer [18] which allowed for these operations. In addition the model changed to include the states of the participants as well as the state of the intruder while still maintaining the old paradigm of unifying against the right hand sides of transition rules in order to generate predecessor states. However, if anything, the model has become more complex, and it still suffers from the most important weaknesses of the original system. There is no systematic way of converting a protocol description into a set of transition rules for the NRL Analyzer. The model checker also relies heavily on the user during the verification much in the same way a theorem prover relies on the user to guide it during the search for a proof. Finally, the algorithms used in the NRL Analyzer are not guaranteed to terminate, and so a limit is placed on the number of recursive calls allowed for some of the model checking routines.

Woo and Lam propose a much more intuitive model for authentication protocols [23]. Their model resembles sequential programming with each participating principal being modelled independently. There is an easy and obvious translation from the common description of a protocol as a set of messages to their model. Their models are also more intuitive because they consider all possible execution traces instead of considering just the set of words obtainable by the intruder. They are concerned with checking for what they call *secrecy* and em correspondence properties. The secrecy property is expressed as a set of words (usually keys) that the intruder is not allowed to obtain. The correspondence properties can express things of the form if principal A finishes a protocol run with principal B, then principal B must have started (participated in) the protocol run with A. However, they do not provide a general logic in which to formalize security properties, nor do they provide an automated tool. Instead they present a set of inference rules with which you can prove correspondence assertions about a model [24]. In addition, the description of their model, while intuitive, is not very precise or formal.

Bolignano presents a model that is almost a middle point between these last two [4]. Like Meadows, Bolignano emphasizes the algebraic properties of the intruder when trying to derive words. The state of the intruder then is the set of words it can generate, while the state of the participants is determined by the values of the variables that correspond to the protocol and their program counters. A number of rules to reason about what information is contained in what messages are provided which can then be used to prove properties about a protocol. In the example given, all properties, including authentication, are given in terms of an invariant that must be proven. Because the invariant must be proven to hold for all protocol steps, this can become unwieldy very quickly.

Other recent work in this area has involved trying to use generic verification tools to verify security protocols. In [16], Lowe uses the FDR model checker for CSP [12] to analyze the Needham-Schroeder Public-Key Authentication Protocol [19]. Lowe succeeded in finding a previously unpublished error in the protocol. The fact that he was able to use a generic model checker is promising as well. Unfortunately, the CSP model for the protocol is far from straightforward. In addition, the model is parameterized by the nonces used by the participants. This means that it only models a single run of the protocol. In order to prove the general protocol correct he must prove a theorem that states that the general protocol is insecure only if this restricted version is insecure.

Leduc and others recently used the LOTOS language [5] and the Eucalyptus tool-box [9] to analyze the Equicrypt protocol [15]. What makes this an interesting case study is the fact that the Equicrypt protocol is a real system currently under design for use in controlling access to multimedia services broadcast on a public channel. They were able to find a couple of security flaws in this proposed system using these generic tools.

Gray and McLean propose encoding the entire protocol in terms of temporal logic [10]. Much like symbolic model checking, they describe the model by giving formulas that express the possible relationships between variable values in the current state and variable values in the next state. This makes their framework more formal than the others, but much more cumbersome as well. They provide a simple example and prove a global invariant for this example. The few subcases they consider are very straightforward but their technique demands very long proofs even for the extremely simple example they present. They argue that their technique could be automated but provide no tool for their system.

Abadi and Gordon propose the spi calculus, an extension of the pi calculus with cryptographic primitives, as another model for describing and analyzing cryptographic protocols [1]. The spi calculus models communicating processes in a way that is very similar to CSP and CCS. The spi calculus provides constructs for output on a channel, input on a channel, restriction, composition, testing for equality, pairs and projections, encryption, decryption and for branching on equality to zero. What sets the spi calculus (and the pi calculus) apart from other calculi is the dynamic nature of the scope of restriction. The restriction operator can be thought of as creating a new name to which only processes within the scope of the restriction operator can refer. However, one of these processes could output this new name outside the scope of the restriction operator allowing another process to refer to it. In the pi calculus, these new names can be thought of as private channels. In the spi calculus, the restriction operator is used to model nonces and keys. So far, protocol models have been verified by comparing to a slightly altered model that is "obviously" correct, and is, therefore, at the same level of abstraction as the protocol model.

A more concrete and complete model is presented by Heintze and Tygar [11]. They view protocols as a set of agents modeled as non-deterministic finite state machines. The actions of a principal who must follow the protocol depend on the local state of that principal and so are in some sense restricted. The actions of adversaries are not restricted by the protocol and hence they are allowed to perform any actions consistent with their current knowledge. (In other words, they cannot send messages that they cannot generate from their current knowledge). Their model also includes a notion of belief, which along with the sequence of sends and receives, defines the local state of a principal. Security is then split into secret-security and time-security. A model is secret-secure if all beliefs are universally valid. In particular if any principal ever believes that a message M is only shared among the principals in S, then it is always the case that if A knows M then $A \in S$. A model is time-secure if all beliefs eventually expire. In other words, if b is a belief held by a principal A at event e then there is an event e' such that b is not held at any event following e'. The authors go on to prove that the questions "Is P secret-secure?" and "Is P time-secure?" are undecidable. While this model does a good job of capturing what one means by "security," the model seems too complex to be used in practice.

3 Intuition

We also propose a model checking scheme for the verification of security protocols and we make use of the same "perfect encryption" assumptions. We propose a very intuitive model which captures the basic idea of message generation and communication. Unlike other systems, where the protocol must be encoded in CSP or in term rewrite rules, in our model, protocol definitions are easily translated into a sequence of commands like SEND, RECEIVE, and NEWNONCE. In fact, it seems clear that this translation could even be done automatically from the simple notation used to describe protocols in the literature as sequences of messages that occur during a run of the protocol.

Once we have a sequence of actions for each of the participants we take their asynchronous composition to get the full model of the protocol. There is one other unspecified participant which we call the intruder. The intruder models an untrusted communication medium as well as any malicious principals. When messages are sent they can always be intercepted by the intruder. The intruder is also allowed to send messages while impersonating a trusted principal. The intruder may even be selected as a participant in a protocol run.

In addition, the intruder will be allowed to compromise temporary secrets, such as session keys, which are generated during the run of the protocol and are not meant to be treated as permanent secrets. Care must be taken, however, because it is unreasonable to allow the intruder to compromise temporary session keys as soon as they are generated. In some sense, the participants should be allowed to make some use of the key before it is allowed to be compromised.

A run of the protocol will then consist of some interleaving of actions from the participants and the intruder. This particular run or *trace* can then be analyzed to determine if the security of the protocol was compromised. In particular we can check if the intruder ever learns a secret which is meant to be permanent or if some principal A believes it has completed a run with principal B, while principal B has not participated in the run. In general, a set of security requirements can be specified in some kind of logic and then the trace can be checked to see if any of these requirements are violated. However, to verify that a protocol is correct, all the possible runs must be checked.

We can think of a trace as an alternating sequence of global states and actions. The global state will consist of the local state of each participant together with some global information like the set of secret information, and which principals have participated in which protocol runs. Since each principal has a finite number of actions it can take at any point in time (typically just one), then the number of possible next states is finite. If we restrict ourselves to a sufficiently large, but still finite number of runs, then the entire state space will be finite and we can do depth-first search of the state space simply checking that no reachable state violates the security specification.

4 The Specification

There are two kinds of properties that we currently are interested in. The first is a kind of secrecy property. We provide the model checker with a set of terms which the intruder is not allowed to obtain. During the verification, we simply check that the intruder does not have possession of any of the terms in this set. This is not as straightforward as it might seem because the information known to the intruder is typically infinite. For example, if the intruder knows a piece of data and a key, it can repeatedly encrypt this data to produce an infinite number of new terms.

The second property is a temporal property that Woo and Lam call *correspondence* [23]. In particular, we are interested in checking that "if principal A believes it has finished a protocol run with principal B, then principal B must have begun a protocol run with principal A." This can be generalized to "if event X occurs, then event Y must have occurred in the past." (We will use Woo and Lam's notation $X \hookrightarrow Y$ to denote this.) However, there is more to this property than a simple temporal relationship. The relation between Y events and X events must be a one-to-one mapping. More formally, the projection of any trace onto X events and Y events must be derivable from the following grammar:

$$S \to SxSy|\epsilon$$

where the terminal symbols x and y represent the events X and Y. In particular, if principal A believes it has completed two protocol runs with principal B, then principal B must have at least begun two protocol runs with principal A. Each end of a protocol run on A's part must be mapped to a separate beginning of a protocol run on B's part.

In order to check for this kind of property, we will augment the global state with counters. For each correspondence property $X \hookrightarrow Y$ we will maintain a separate counter which will keep track of the difference between the number of Y events and X events. If this counter ever turns negative (i.e. there are more X events than Y events) then the correspondence property will be violated at that point (there will be no one-to-one mapping from X events to Y events). Conversely, as long as the counter never goes negative there is always a one-to-one mapping from X events to Y events.

5 Messages

Typically, the messages exchanged during the run of a protocol are built up using pairing and encryption from smaller submessages. The smallest such submessages (i.e. they contain no submessages themselves) are called *atomic messages*. There are four types of *atomic messages*.

- *Keys* are used to encrypt messages. We make the "perfect encryption" assumption, which states that the only way to obtain the plaintext from an encrypted message is by using the appropriate decryption key. Keys have the property that every key k has an inverse k^{-1} such that for all messages m, $\{\{m\}_k\}_{k^{-1}} = m$. (Note that for symmetric cryptography the decryption key is the same as the encryption key, so $k = k^{-1}$.)
- *Principal names* are used to refer to the participants in a protocol.
- *Nonces* are randomly generated numbers. The intuition is that since they are randomly generated, any message containing a nonce can be assumed to have been generated after the nonce was generated. (It is not an "old" message.)
- Data which plays no role in how the protocol works but which is intended to be communicated between principals.

Let $\mathcal{A}$ denote the space of *atomic messages*. The set of all messages $\mathcal{M}$ over some set of atomic messages $\mathcal{A}$ is defined inductively as follows:

- If $a \in \mathcal{A}$ then $a \in \mathcal{M}$. (Any *atomic message* is a message.)
- If $m_1 \in \mathcal{M}$ and $m_2 \in \mathcal{M}$ then $m_1 \cdot m_2 \in \mathcal{M}$. (Two messages can be paired together to form a new message.)
- If $m \in \mathcal{M}$ and key $k \in \mathcal{A}$ then $\{m\}_k \in \mathcal{M}$. (A message M can be encrypted with key k to form a new message.)

Because keys have inverses, we take this space modulo the equivalence $\{\{m\}_k\}_{k^{-1}} = m$. It is also important to note that we make the following perfect encryption assumption. The only way to generate $\{m\}_k$ is from m and k. In other words, there do not exist messages m, m_1, and m_2 and key k such that $\{m\}_k = m_1 \cdot m_2$, and $\{m\}_k = \{m'\}_{k'}$ implies $m = m'$ and $k = k'$.

Let $B \subseteq \mathcal{M}$ be a subset of messages. The closure of B (denoted $\overline{B}$), representing the set of everything that can be derived from B, is defined by the following rules:

1. If $m \in B$ then $m \in \overline{B}$.
2. If $m_1 \in \overline{B}$ and $m_2 \in \overline{B}$ then $m_1 \cdot m_2 \in \overline{B}$. (pairing)
3. If $m_1 \cdot m_2 \in \overline{B}$ then $m_1 \in \overline{B}$ and $m_2 \in \overline{B}$. (projection)
4. If $m \in \overline{B}$ and key $k \in \overline{B}$ then $\{m\}_k \in \overline{B}$. (encryption)
5. If $\{m\}_k \in \overline{B}$ and key $k^{-1} \in \overline{B}$ then $m \in \overline{B}$. (decryption)

6 The Model

We now define the model formally by describing how the overall global state and the individual principal local states are defined as well as by describing how actions update the state. The model consists of the asynchronous composition of a set of named, communicating processes, each augmented with a local store in which to keep track of the current information it "knows", and with a set of bindings for the variables appearing in the process. Each principal involved in the protocol is modelled as one of these processes and is described by a sequence of actions it is to perform and by the initial state of its local store. The initial state of the bindings is assumed to be empty. One process, the intruder, is not completely specified. Only the initial state of its local store is given and it is allowed to perform any "realistic" actions. For example, the intruder is not allowed to decrypt messages with a key it does not possess and it is not allowed to send messages that it cannot create with the information in its local store. But it is allowed to receive and send messages arbitrarily, possibly intercepting messages intended for other principals or possibly impersonating a trusted principal.

More formally, each principal is modelled as a 4-tuple $\langle N, p, I, B\rangle$, where:

- $N \in$ *names* is the name of the principal.
- p is a process (similar in style to CSP) given as a sequence of actions to be performed.
- $I \subseteq \mathcal{M}$ is a set of all messages known (which can be produced) by the principal. $\mathcal{M}$ is the set of all possible messages. Typically I will be infinite and in particular, it is closed under encryption, decryption, pairing (concatenation), and projection. For example, if $m, k \in I$ then $\{m\}_k \in I$. For some set of messages J, we will use $\overline{J}$ to denote the closure of J under these operations.
- $B : \mathit{vars}(p) \to I$, where $\mathit{vars}(p)$ is the set of variables appearing in the process p, is a set of bindings.

The global state is then maintained as the composition of the participating principals, along with the intruder process, a list of permanent secrets, a list of temporary secrets, and a set of counters indexed by the pairs of principals participating in protocol runs. More formally, the global state is a 5-tuple $\langle \Pi, C_i, C_r, S_s, S_t\rangle$, where:

- Π is the product of the the individual principals and the intruder process. This product is asynchronous, yielding an interleaving semantics, with the restriction that processes synchronize on messages.
- $C_i : \mathit{names} \times \mathit{names} \to \mathbb{N}$ gives the difference between the number of times some principal with name A has begun initiating a protocol with some other principal with name B and the number of times B has finished responding to principal A. If a counter ever gets a negative value this means that B has finished responding in a protocol with A (i.e. believes A has participated in the protocol) without A having taken part in the protocol.
- $C_r : \mathit{names} \times \mathit{names} \to \mathbb{N}$ gives the difference between the number of times some principal named A has begun responding to some other principal named B and the number of times B has finished initiating a protocol with A. If a counter ever gets a negative value this means that B has finished initiating a protocol with A (i.e. believes A has participated in the protocol) without A having taken part in the protocol.
- $S_s \subseteq \mathcal{M}$ is a set of messages that are are considered safe secrets. These are the set of words that the intruder is never allowed to know. This set remains constant and usually includes things like the private keys that principals use to communicate with a server.
- $S_t \subseteq \mathcal{M}$ is a set of messages that are are considered temporary secrets. This is the set of new secrets generated during the run of the protocol. These are secrets which we assume the intruder may be able to discover by some outside means, but which the protocol should not reveal, such as session keys.

The specific actions that a principal may perform can be divided into internal actions and communication actions. The internal actions are performed asynchronously. Any principal is allowed to perform an internal action and interleaving is used to model all possible behaviors when multiple principals can make a transition. We define a transition relation $\to$ between principals such that $\mathbf{A} \to \mathbf{B}$ if and only if principal $\mathbf{A}$ can take an action and become a principal that behaves like $\mathbf{B}$.

Communication actions consist of send and receive actions. Each receive action can potentially change the principal's local store, reflecting any new information it has "learned." Communication actions can only occur in pairs and both principals make a transition simultaneously. These communication actions are also interleaved with the possible actions of other automata.

In order for a communication action to take place, the message being sent must *unify* with the message being received. A message *s-msg* from principal $\mathbf{A} = \langle A, p, I_A, B_A\rangle$ unifies with a message *r-msg* from principal $\mathbf{B} = \langle B, q, I_B, B_B\rangle$, if there exist a substitution $\sigma_B : \mathit{vars}(q) \to I_A$ extending B_B ($B_B \subseteq \sigma_B$), such that $B_A(\textit{s-msg}) = \sigma_B(\textit{r-msg})$. If the messages unify, then the following transitions can be taken:

$$\begin{array}{rcl}\langle A,\ \textsc{send}(\textit{s-msg}).p', I_A, B_A\rangle & \rightarrow & \langle A, p', I_A, B_A\rangle \\ \langle B,\ \textsc{receive}(\textit{r-msg}').q', I_B, B_B\rangle & \rightarrow & \langle B, q', I'_B, \sigma_B\rangle\end{array}$$

where $I'_B = \overline{I_B \cup \sigma_B(\textit{r-msg})}$. Because we require that *s-msg* unify with *r-msg*, if there is already a pair (*var*, *val*) in B for some *var* appearing in *r-msg*, then the corresponding value in *s-msg* must be *val*. Thus the updates to B only add new bindings and never change previous bindings.

For the most part internal actions are used to create or discover new information. For example, NEWNONCE is used to create a nonce. Nonces are globally distinct, and each NEWNONCE action creates a nonce that has not appeared up to that point in the protocol. The new nonce is added to the principal's local store. NEWSECRET works similarly, except that this is supposed to model generating a new session key which can then be used to encrypt messages. More formally:

$$\begin{array}{rcl}\langle A, \textsc{newnonce}(\textit{var}).p', I, B\rangle & \rightarrow & \langle A, p', I', B'\rangle \\ \langle A, \textsc{newsecret}(\textit{var}).p', I, B\rangle & \rightarrow & \langle A, p', I', B'\rangle\end{array}$$

where in both cases, if *val* is the new value generated by the action, then $I' = \overline{I \cup \textit{val}}$ and $B' = B[\textit{var} \leftarrow \textit{val}]$. If the action was a NEWSECRET action, then the S_t is updated in the global state as well to $S'_t = S_t \cup \textit{val}$.

Additionally, the intruder is allowed to perform a GETSECRET action which it can use to acquire a secret previously generated by a principal using NEWSECRET. This models the possibility of session keys being compromised. It allows us to have two classes of secrets, those which we assume to be "permanent" like a private key between a server and a trusted principal, and those secrets which are "temporary" such as session keys. We need to allow the intruder to obtain session keys in order to allow for the possibility of replay attacks which would allow the intruder to establish an old compromised key as a session key. However, we also need to restrict the the usage of GETSECRET or else the intruder would be allowed to compromise a session key immediately after it is generated and before it is ever used. For this reason, we only allow the intruder to perform a GETSECRET action to compromise a key which has already been established or used in a protocol. Formally,

$$\langle Z, \textsc{getsecret}.p', I, B\rangle \quad \rightarrow \quad \langle Z, p', I', B\rangle$$

where for some $\textit{val} \in S_t, I' = \overline{I \cup \textit{val}}$ and in the global state S_t is updated to $S'_t = S_t - \{\textit{val}\}$.

Finally, we have four special actions BEGINIT, ENDINIT, BEGRESPOND, and ENDRESPOND. These are used to mark the beginning and the end of a principal's participation in a protocol. We use them to guarantee that if the principal named A finishes the protocol (performs ENDINIT(B)) then the principal named B has participated in the protocol (performed BEGRESPOND(A)). We do this by maintaining counters for each pair of principals participating in a protocol. More formally,

$$\langle A, \textsc{beginit}(B).p', I_A, B_A\rangle \quad \rightarrow \quad \langle A, p', I_A, B_A\rangle$$

and we update the global state by setting the new value of $C_i(A, B)$:

$$C'_i(A, B) = \begin{cases} C_i(A, B) + 1 & \text{if } C_i(A, B) \text{ is defined} \\ 1 & \text{otherwise} \end{cases}$$

Similarly,

$$\langle B, \textsc{endrespond}(A).p', I_B, B_B\rangle \quad \rightarrow \quad \langle B, p', I_B, B_B\rangle$$

and we update the global state by setting the new value of $C_i(A, B)$:

$$C_i'(A,B) = \begin{cases} C_i(A,B) - 1 & \text{if } C_i(A,B) > 0 \\ \mathit{error} & \text{otherwise} \end{cases}$$

The definitions for BEGRESPOND and ENDINIT are identical except that C_r is updated in the global state instead of C_i.

The GETSECRET action may only be performed by the intruder, while the rest of the actions may be performed by any principal. The actions a particular honest principal may make are restricted to the sequence of actions p that represent its role in the protocol. The intruder has no such restriction and is allowed to make any action at any time, provided that if it performs a SEND action with message m, it must be the case that $m \in \overline{I_Z}$.

Recall that a trace is an alternating sequence of global states and actions and that we are interested in checking all possible traces. Clearly, there are a finite number of next states for each of the participants. In addition, while the intruder can generate an infinite number of messages, it is only allowed to send a finite number because each SEND much match with a RECEIVE. Since the there are a finite number of possible next states, we only consider a finite number of runs, we can perform a depth first search of the state space to generate all possible traces. We then check that no reachable state violates the security specification. Pseudocode for this algorithm can be found in figure 1.

proc *DFS* (*global-state*)
push(*global-state*,S)
while (**not** *empty*(S)) **do**
    $\langle \Pi, C_i, C_r, S_s, S_t \rangle$ = *pop*(S)
    if $C_i(x, y) < 0$ for some x and y **or**
        $C_r(x, y) < 0$ for some x and y **or**
        $s \in I_Z$ for some $s \in S_s \cup S_t$
            /* *where* I_Z *is the intruder's information in* Π. */
    then *report-error*
    L = *next-states*($\langle \Pi, C_i, C_r, S_s, S_t \rangle$)
    for each $l \in L$ *push*(S, l)

Figure 1: Model-checking algorithm

The remaining detail is how to maintain the local stores for the principals. The local store is accessed in three places. First, if principal $\langle A, p, I_A, B_A \rangle$ sends a message m, then we must insure that $m \in \overline{I_A}$. Second, if the principal receives message m, then we must update I_A to $I_A' = \overline{I_A \cup m}$. Finally, we check every global state to see if $s \in I_Z$ for some $s \in S_s \cup S_t$, where I_Z is the intruder's local store. It turns that these local stores are infinite because of the closure operation. However, we never really need to compute the entire closure; we need only determine if a particular message is in the closure. So it suffices to represent the infinite set with a finite set of "generators." This is the topic of the next section.

7 Normalized Derivations

Intuitively speaking, if B represents some set of information that is known by a principal, then the principal also knows (can generate) all the information in $\overline{B}$. In general $\overline{B}$ is an infinite set; however, we usually are not interested in the set of everything that a principal knows, but instead whether or not a specific message $x \in \mathcal{M}$ can be generated by a principal. This leads us to the following definition.

Let $x \in \overline{B}$ be a message. A *derivation* of x from B is an alternating sequence of sets of messages and rule instances written as follows:

$$B_0 \stackrel{R_0}{\rightarrow} B_1 \stackrel{R_1}{\rightarrow} \cdots B_{k-1} \stackrel{R_{k-1}}{\rightarrow} B_k$$

where:

- $B = B_0$
- $x \in B_k$
- Each rule instance R_i is written as $\langle I_i, N_i, O_i \rangle$ where:
  - $I_i \subseteq B_i$
  - $B_{i+1} = B_i \cup O_i$
  - N_i is one of the closure rules for $\overline{B}$ such that I_i satisfies the premise of the rule and O_i is the corresponding conclusion.

For example, let $B = \{\{a\}_k \cdot b, k^{-1}\}$. We derive $x = a \cdot b$ as follows:

1. $B_0 = B = \{\{a\}_k \cdot b, k^{-1}\}$
2. $R_0 = \langle \{\{a\}_k \cdot b\}, 3, \{\{a\}_k, b\} \rangle$
3. $B_1 = \{\{a\}_k \cdot b, k^{-1}, \{a\}_k, b\}$
4. $R_1 = \langle \{\{a\}_k, k^{-1}\}, 5, \{a\} \rangle$
5. $B_2 = \{\{a\}_k \cdot b, k^{-1}, \{a\}_k, b, a\}$
6. $R_2 = \langle \{a, b\}, 2, \{a \cdot b\} \rangle$
7. $B_3 = \{\{a\}_k \cdot b, k^{-1}, \{a\}_k, b, a, a \cdot b\}$ which contains x

We would now like to introduce the notion of a *normalized derivation*, but first we must introduce the notion of *shrinking rules* and *expanding rules* by defining a *metric* $\mu : \mathcal{M} \to \mathbb{N}$. We then define a *shrinking rule* to be a rule such that for every instance of the rule $\langle I, N, O \rangle$ we have:

$$\max_{m \in I} \mu(m) > \max_{m \in O} \mu(m)$$

Analogously, an *expanding rule* is a rule for which every instance $\langle I, N, O \rangle$ we have:

$$\max_{m \in I} \mu(m) < \min_{m \in O} \mu(m)$$

We can now define a *normalized derivation* as follows:

$$B_0 \xrightarrow{R_0} B_1 \xrightarrow{R_1} \cdots B_{k-1} \xrightarrow{R_{k-1}} B_k$$

is a normalized derivation if and only if for all $0 \leq i < k, N_i$ is an expanding rule implies N_j is an expanding rule for all $i < j \leq k$. In other words, all shrinking rules appear to the left of all expanding rules. Recall that in our notation, R_i is the rule instance $\langle I_i, N_i, O_i \rangle$,

For example, in our model, we will define our metric μ inductively as follows:

- $\mu(a) = 1$ for all $a \in \mathcal{A}$
- $\mu(m_1 \cdot m_2) = \mu(m_1) + \mu(m_2)$
- $\mu(\{m\}_k) = \mu(m) + 1$

Note that $\mu(m)$ is well defined when $m = \{m_1\}_{k_1} = \{m_2\}_{k_2}$, because the perfect encryption assumption implies that $m_1 = m_2$ and $k_1 = k_2$. In the case $m = m_1 \cdot m_2 = m_1' \cdot m_2'$ either m_1 is a substring of m_1' or m_1' is a substring of m_1. Without loss of generality, assume $m_1 = m_1' \cdot b$. Then it must be the case that $m_2' = b \cdot m_2$ because we have $m = m_1 \cdot m_2 = m_1' \cdot b \cdot m_2 = m_1' \cdot m_2'$. Therefore

$$\mu(m) = \mu(m_1 \cdot m_2) = \mu(m_1' \cdot b \cdot m_2) = \mu(m_1' \cdot m_2').$$

The message derivation rules from section /refsect:messages can now be categorized. With these definitions, rules 3 and 5 are shrinking rules and rules 2 and 4 are expanding rules.

We now show that in our model, there is a derivation of x from B if and only if there is a normalized derivation of x from B. First we need the following lemma.

Lemma 1: Let $B_0 \stackrel{R_0}{\to} B_1 \stackrel{R_1}{\to} B_2$ be a derivation of length 2 such that N_0 is an expanding rule and N_1 is a shrinking rule. There there exists a derivation $B'_0 \stackrel{R'_0}{\to} B'_1 \stackrel{R'_1}{\to} \cdots B'_{k-1} \stackrel{R'_{k-1}}{\to} B'_k$ such that

1. $N'_1, \ldots, N'_{k-1}$ are expanding rules.
2. $B_0 = B'_0$
3. $B_2 \subseteq B'_k$

Proof:

Case $N_0 = 2$ and $N_1 = 3$:

Let $R_0 = \langle \{m_1, m_2\}, 2, \{m_1 \cdot m_2\} \rangle$ and $R_1 = \langle \{m'_1 \cdot m'_2\}, 3, \{m'_1, m'_2\} \rangle$

Case I: $m'_1 \cdot m'_2 \neq m_1 \cdot m_2$ or $m'_1 \cdot m'_2 \in B_0$

In either case, $m'_1 \cdot m'_2 \in B_0$, and the new derivation is

$$\begin{array}{rcl} R'_0 & = & R_1 \\ R'_1 & = & R_0 \end{array}$$

It is clear that $B'_2 = B_2$.

Case II: $m'_1 \cdot m'_2 = m_1 \cdot m_2$ and $m'_1 \cdot m'_2 \notin B_0$

If we also have $m'_1 = m_1$ and $m'_2 = m_2$, then $m'_1, m'_2 \in B_0 \subseteq B_1$. Therefore $B_2 = B_1$ and we let the new derivation consist only of

$$R'_0 \;=\; R_0$$

Otherwise, we must have that either m_1 is a substring of m'_1 or m'_1 is a substring of m_1. Without loss of generality, assume $m_1 = m'_1 \cdot b$. Then it must be the case that $m'_2 = b \cdot m_2$ because we have $m = m_1 \cdot m_2 = m'_1 \cdot b \cdot m_2 = m'_1 \cdot m'_2$. Then the new derivation becomes:

$$\begin{array}{rcl} R'_0 & = & \langle \{m_1\}, 3, \{m'_1, b\} \rangle \\ R'_1 & = & \langle \{b, m_2\}, 2, \{m'_2\} \rangle \\ R'_2 & = & \langle \{m_1, m_2\}, 2, \{m_1 \cdot m_2\} \rangle \end{array}$$

And we have that

$$B'_3 = B_0 \cup \{m'_1, b\} \cup \{m'_2\} \cup \{m_1 \cdot m_2\} = B_2 \cup \{b\}$$

Case $N_0 = 2$ and $N_1 = 5$:

Let $R_0 = \langle \{m_1, m_2\}, 2, \{m_1 \cdot m_2\} \rangle$ and $R_1 = \langle \{\{m\}_k, k^{-1}\}, 5, \{m\} \rangle$

One of our assumptions about encryption is that given m, the only way to generate $\{m\}_k$ is by knowing m and k and using the encryption algorithm. Therefore there are no m_1 and m_2 such that $m_1 \cdot m_2 = \{m\}_k$. So, in this case, $\{m\}_k \in B_0$ and the new derivation becomes

$$\begin{array}{rcl} R'_0 & = & R_1 \\ R'_1 & = & R_0 \end{array}$$

It is clear that $B_2 = B'_2$.

Case $N_0 = 4$ and $N_1 = 3$:

Let $R_0 = \langle\{m, k\}, 4, \{\{m\}_k\}\rangle$ and $R_1 = \langle\{m_1 \cdot m_2\}, 3, \{m_1, m_2\}\rangle$

Again, since we can't have $m_1 \cdot m_2 = \{m\}_k$, we must have that $m_1 \cdot m_2 \in B_0$ and the new derivation becomes

$$\begin{aligned} R_0' &= R_1 \\ R_1' &= R_0 \end{aligned}$$

Again, $B_2 = B_2'$.

Case $N_0 = 4$ and $N_1 = 5$:

Let $R_0 = \langle\{m, k\}, 4, \{\{m\}_k\}\rangle$ and $R_1 = \langle\{\{m'\}_{k'}, k'^{-1}\}, 5, \{m'\}\rangle$

Case I: $\{m'\}_{k'} = \{m\}_k$
In this case, we also have $m' = m$ and $k' = k$, therefore $B_1 = B_2$ and so the new derivation is:

$$R_0' = R_0$$

Clearly, $B_1' = B_1 = B_2$.

Case II: $\{m'\}_{k'} \neq \{m\}_k$
It must be the case that $\{m'\}_{k'} \in B_0$ so the following is a valid derivation:

$$\begin{aligned} R_0' &= R_1 \\ R_1' &= R_0 \end{aligned}$$

It is clear that $B_2' = B_2$.

Theorem 2: Let $B \subseteq \mathcal{M}$ be a set of messages. Then $x \in \overline{B}$ if and only if x has a normalized derivation from B.

Proof: If x has a normalized derivation from B then clearly this is a derivation and by definition $x \in \overline{B}$.
For the other direction, let $x \in \overline{B}$. Then there exists some derivation

$$\Gamma = B_0 \overset{R_0}{\to} B_1 \overset{R_1}{\to} \cdots B_{k-1} \overset{R_{k-1}}{\to} B_k$$

such that $x \in B_k$. Let $S = \{i | R_i$ is a shrinking rule and $\exists j < i$ such that R_j is an expanding rule $\}$. If S is empty, then Γ is a normalized derivation and we are done. Otherwise, we can induct on the size of S. Let $r = \min S$. By repetitively using Lemma 1, we can move R_r to the left, until either it is the leftmost rule, or it is immediately to the right of another shrinking rule. Since the original derivation is finite and since each time we apply Lemma 1, rule R_r moves one slot to the left, we need apply Lemma 1 only a finite number of times. If R_r becomes the leftmost rule, then clearly there are no expanding rules to the left of R_r. If R_r is now immediately to the right of another shrinking rule R_s, then there are still no expanding rules to the left of R_r because then there would be an expanding rule to the left of R_s in the original derivation and so $s \in S$ and $s < r$ contradicting the minimality of r. Now we have a new derivation of x, Γ', which is still finite. Since the application of Lemma 1 does not add any new shrinking rules, S', the new S, satisfies $S' = S - \{r\}$. Furthermore $|S'| = |S| - 1$, so by the inductive hypothesis we can transform Γ' into a normalized derivation of x.

Corollary 3: Given $x \in \mathcal{M}$ and $B \subseteq \mathcal{M}$, determining if $x \in \overline{B}$ is decidable.

Proof: By Theorem 2, $x \in \overline{B}$ if and only if x has a normalized derivation from B. We therefore try to find a normalized derivation or show that none exists. First we repeatedly apply shrinking rules to $B = B_0$ creating new sets B_i. Since there are a finite number of rules, each rule creates a finite number of new words, each smaller (by the metric μ) than each of the words used as an input to the rule, and B_0 is finite to begin, there are only a finite number of B_i's and hence we only apply shrinking rules a finite number of times. Let us call this final set B_s. Since B_s is the result of repeatedly applying all possible shrinking rules to B, x has a normalized derivation from B if and only if it has a derivation from B_s which uses only expanding rules. Furthermore, the length of a minimal derivation of x from B_s is bounded by $\mu(x)$ since each expanding rule creates a words that are longer than the words used as inputs to the rule. Since there are a finite number of expanding rules and B_s is itself finite, we can simply try all possible sequences of expanding rules of length less than or equal to $\mu(x)$ in a finite number of steps. Therefore, this whole algorithm is guaranteed to terminate.

In the proof of Lemma 1, the majority of cases displayed a kind of independence of rules. Intuitively, independence means that applying one rule does not increase the set of things that can be derived using the other rule. More formally, a shrinking rule s is *independent* of an expanding rule e if for each pair of instances $\langle I_s, s, O_s\rangle$ and $\langle I_e, e, O_e\rangle$ we have one of the following:

1. $O_e \cap I_s = \phi$: The output of the expanding rule cannot be used as input to the shrinking rule. This is the case for pairing and decryption and for encryption and projection.
2. $O_s \subseteq I_e$: The information gained by applying the shrinking rule was already present when applying the expanding rule. This could be the case when for encryption and decryption using the same key.

Note that this property applied to almost all cases of Lemma 1 and that the only real work in proving Lemma 1 came from the case of the pairing rule and projection rule because these are not independent. The other pairs of rules were independent because of the "perfect encryption assumption." In general, this exchanging property (Lemma 1) need only be shown for pairs of rules that are not independent.

8 Information Algorithms

While Corollary 3 proves the decidability of determining if $x \in \overline{B}$, it is an extremely inefficient algorithm. In particular, enumerating all sequences of expanding rules of length $\mu(x)$ will yield exponential complexity. In practice however, we can search for a derivation of x from B_s by using the structure of x. Specifically, we have the following theorems:

Theorem 4: $m_1 \cdot m_2 \in \overline{B_s}$ if and only if $m_1 \cdot m_2 \in B_s$ or $m_1 \in \overline{B_s}$ and $m_2 \in \overline{B_s}$.

Proof: Assume $m_1 \cdot m_2 \in \overline{B_s}$ and $m_1 \cdot m_2 \notin B_s$, then $m_1 \cdot m_2$ must be in $\overline{B_s}$ because of an expanding rule. By assumption, $m_1 \cdot m_2 \notin B_s$. To show that $m_1 \cdot m_2 \in \overline{B_s}$ can be derived from B_s without using a shrinking rule we take a derivation of $m_1 \cdot m_2 \in \overline{B_s}$, Γ, and use theorem 2 to get a normalized derivation Γ'. Now either the shrinking rules appearing in Γ' are redundant (i.e. they don't add any new words and so can be removed from the derivation) or we contradict the fact that B_s was created by applying all possible shrinking rules to B. In either case the remainder of the derivation (and there must be some remainder since we assume that $m_1 \cdot m_2 \notin B_s$) must consist of expanding rules. In particular the last rule used in the derivation must be an expanding rule and the only way that could be the case is if it is rule 2 which would require as its premise $m_1 \in \overline{B_s}$ and $m_2 \in \overline{B_s}$.

Now assume that $m_1 \cdot m_2 \in B_s$ or $m_1 \in \overline{B_s}$ and $m_2 \in \overline{B_s}$. Then it is clear by either rule 1 or rule 2 that $m_1 \cdot m_2 \in \overline{B_s}$.

Theorem 5: $\{m\}_k \in \overline{B_s}$ if and only if $\{m\}_k \in B_s$ or $m \in \overline{B_s}$ and $k \in \overline{B_s}$.

Proof: Analogous to the previous theorem.

Putting all these together yields the basis for our search algorithm. As our set of known messages increases, we repeatedly apply shrinking rules and removing "redundant messages" until we get a set of "basic" messages, B_s, to which we cannot apply any shrinking rules. By redundant messages, we mean messages that can be generated from the other messages in the set using expanding rules. For example, when we apply rule 3 to get m_1 and m_2 from $m_1 \cdot m_2$, we also remove $m_1 \cdot m_2$ from B_s. However, when applying rule 5 we must be careful; when we generate m from $\{m\}_k$ and k^{-1} we cannot remove $\{m\}_k$ from B_s unless $k \in B_s$. Pseudocode for this algorithm is given in figure 2.

We now consider the complexity of inserting a new message m into our current set of information B_s and generate a new set of information B_s'. The only time there is any interaction between previously known messages in B_s and m is when we try to apply the decryption rule. The message m can have at most $|m|$ encryptions. For each encryption, we scan B_s looking for the inverse key for a total of $|B_s||m|$ time. Analogously, m could contain at most $|m|$ keys. For each key, we must check each element of B_s to see if it

```
function add(I, m)

for each i ∈ I
      if i = {x}_y and y^{-1} = m
            then I = add(I, x)
      if y ∈ I then I = I − i
if m = x · y
      then return add(add(I, x),y)
if m = {x}_y and y^{-1} ∈ I
      then if y ∈ I
            then return add(I, x)
            else return add(I ∪ m, x)
return I ∪ m
```

Figure 2: Augmenting the intruder's knowledge

can now be decrypted. Again, this takes at most $|B_s||m|$ time. However, the newly decrypted message could again be decrypted. The number of iterations is bounded by $|B_s|$; therefore, the total time to generate B'_s, is bounded by $O(|B_s|^2|m|)$ and the size of B'_s is bounded by $O(|B_s|^2)$.

We know that any words in $\overline{B_s}$ can be derived using only expanding rules. When we search to see if a word w is known, we can use theorems 4 and 5 to break it down into smaller pieces which can then be searched recursively. For example, if $w \notin B_s$ and $w = \{m\}_k$, then theorem 5 tells us that $w \in \overline{B_s}$ only if $m \in \overline{B_s}$ and $k \in \overline{B_s}$. Pseudocode for this algorithm is given in figure 3.

```
function in(I, m)

if m ∈ I
      then return true
if m = x · y
      then return in(I, x) and in(I, y)
if m = {x}_y
      then return in(I, x) and in(I, y)
return false
```

Figure 3: Searching the intruder's knowledge

When searching for a derivation of w from B_s we first check to see if $w \in B_s$. This costs at most B_s time. If not, we break down w into two smaller pieces and recursively check those peices. The total number of recursive calls is bounded by the number of operations making up w, which is in turn bounded by $|w|$. Thus the total time to check if $w \in \overline{B_s}$ is bounded by $O(|B_s||w|)$.

9 Verification Example

We now consider an example to illustrate how the model checker works. We consider the simplified Needham-Schroeder protocol analyzed by Lowe [16] given below:

1. $A \to B : A.B.\{N_a.A\}_{K_B}$

2. $B \rightarrow A : B.A.\{N_a.N_b\}_{K_A}$

3. $A \rightarrow B : A.B.\{N_b\}_{K_B}$

Here A is the initiator and B is the responder. A selects a nonce N_a and sends it along with its name encrypted with B's public key to B. B uses its private key to decrypt this message and obtain N_a. Now B generates its own nonce N_b and sends it along with N_a encrypted with A's public key to A. A uses its private key to decrypt this message and returns N_b to B encrypted with B's public key. B then uses its private key to verify that it has just received the nonce sent earlier.

In order to use our model checker, we first isolate which actions are performed by A and which actions are performed by B. We then write a short sequence of actions which make up each participant's role in the protocol. The process description for principal A can be found in figure 4. The description for principal B is similar. All that remains is to specify the initial state of each principal's local store. Each principal, including the intruder, knows the names of all three principals. Each principal also knows the public key of each of the three principals. Finally, each principal knows it's own private key. Figure 5 lists the initial contents of the intruder's local store which consists of the names of the three principals, all three public keys and it's own private key.

```
((beginit (*p-var* b))
 (newnonce (*var* na))
 (send (*var* b)
       (concat a
               (*var* b)
               (encrypt (pubkey (*var* b)) (concat (*var* na) a))))
 (receive (*var* b)
          (concat (*var* b)
                  a
                  (encrypt (pubkey a) (concat (*var* na) (*var* nb)))))
 (send (*var* b)
       (concat a
               (*var* b)
               (encrypt (pubkey (*var* b)) (*var* nb))))
 (endinit (*var* b)))
```

Figure 4: Process description for the initiator

```
(a b *intruder* (pubkey a) (pubkey b)
 (pubkey *intruder*) (privkey *intruder*))
```

Figure 5: The intruder's initial knowledge

The result of the verification attempt can be found in figure 6. In just a few seconds, the model checker finds a violation of the security specification and generates a counter-example. Figure 7 provides an easier to read description of the attack. The sequence of messages for two runs of the protocol (α and β) are provided. The notation $I(A)$ is meant to convey either I impersonating A if on the left of the arrow, or I intercepting a message meant for A if on the right of the arrow.

If we examine the counter-example we can see what has happened. A initiates a protocol run with the intruder. The intruder initiates a protocol run with B impersonating A and using the same nonce that A used with the intruder. When B responds, the intruder forwards this message to A. This message has the format that A is expecting, namely its own nonce and a new nonce encrypted with A's public key. A then replies back to the intruder with B's nonce encrypted with the intruder's public key. The intruder can use its private key to decrypt this and it can now return B's nonce encrypted with B's public key. When B receives this message, the protocol run is complete and B believes it has finished a protocol run with A while A does not have the corresponding belief that it has initiated a protocol run with B.

The above analysis is most easily seen in figure 7 by observing the following relationship between the α run and the β run:

- The role of A in α is played by I in β.
- The role of I in α is played by B in β.
- Each message in β can be obtained from the corresponding message in α by replacing every occurance of I with B.

Therefore, the β run is identical to the α run except that B plays the role of the responder and I impersonating A has played the role of the initiator.

```
"Lack of correspondence"
(B (BEGRESPOND A))
(A (BEGINIT *INTRUDER*))
(A ((NEWNONCE (*VAR* NA)) (*NONCE* 245)))
(A
 (CONCAT A *INTRUDER*
  (ENCRYPT (PUBKEY *INTRUDER*) (CONCAT (*NONCE* 245) A)))
 INTRUDER)
(INTRUDER (CONCAT A B (ENCRYPT (PUBKEY B) (CONCAT (*NONCE* 245) A))) B)
(B ((NEWNONCE (*VAR* NB)) (*NONCE* 260)))
(B
 (CONCAT B A (ENCRYPT (PUBKEY A) (CONCAT (*NONCE* 245) (*NONCE* 260))))
 INTRUDER)
(INTRUDER
 (CONCAT *INTRUDER* A
  (ENCRYPT (PUBKEY A) (CONCAT (*NONCE* 245) (*NONCE* 260))))
 A)
(A (CONCAT A *INTRUDER* (ENCRYPT (PUBKEY *INTRUDER*) (*NONCE* 260)))
 INTRUDER)
(A (ENDINIT *INTRUDER*))
(INTRUDER (CONCAT A B (ENCRYPT (PUBKEY B) (*NONCE* 260))) B)
```

Figure 6: Verification Result

$$
\begin{array}{lcccl}
\alpha 1. & A & \to & I & : & A.I.\{N_a.A\}_{K_I} \\
\beta 1. & I(A) & \to & B & : & A.B.\{N_a.A\}_{K_B} \\
\beta 2. & B & \to & I(A) & : & B.A.\{N_a.N_b\}_{K_A} \\
\alpha 2. & I & \to & A & : & I.A.\{N_a.N_b\}_{K_A} \\
\alpha 3. & A & \to & I & : & A.I.\{N_b\}_{K_I} \\
\beta 3. & I(A) & \to & B & : & A.B.\{N_b\}_{K_B}
\end{array}
$$

Figure 7: Attack on Needham-Schroeder Protocol

Lowe suggests fixing the protocol by changing the second message so that the new protocol is as follows:

1. $A \to B : A.B.\{N_a.A\}_{K_B}$

2. $B \to A : B.A.\{N_a.N_b.B\}_{K_A}$

3. $A \to B : A.B.\{N_b\}_{K_B}$

When we try to verify this protocol, like Lowe, we find no attack in a single run of the protocol. Because no attack was found, the entire exhaustive search of the state space is performed and so the verification process takes a bit longer, but it still completed in under a minute.

10 Conclusion

Our model checker provides a number of advantages over other formalisms. The way we model a protocol is very intuitive. We simply list the sequence of actions that each participant takes in the protocol. Unlike systems based on logics, we need not interpret the beliefs that each message is meant to convey, and we can generate counterexamples when an error is found. Unlike term rewriting approaches, we need not construct a set of rewrite rules to model how an intruder can manipulate participants to generate new messages. We simply model the protocol as a set of programs, one for each participant in the protocol. Because we separate the algorithms that maintain the intruder's knowledge from the state exploration algorithms, we also never need to encode the intruder for our models.

The prototype model checker described here has successfully discovered previously published errors in protocols. When run on correct protocols, the model checker takes a bit longer because it ends up exploring the entire reachable state space, but for the examples investigated so far, the system still terminates in about a minute. We are confident that this kind of exhaustive simulation is a feasible and useful technique for verifying security protocols. However, there are still many extensions that can be investigated and implemented as well as additional experiments to be carried out.

Despite that fact that there is a simple and straightforward translation from protocol descriptions in the literature into our modelling language, this process is tedious and prone to error. We are currently developing a better interface that would allow protocols to be specified exactly the same way they are specified in the literature. We are also working on defining a logic in which to specify the properties we are interested in checking. We are investigating how to add other message operations such as XOR and encryption with non-atomic keys. While these extensions should be possible, it is not clear how these additions will affect the efficiency of our decision procedure for message derivations.

Efficiency is also an important concern. Currently, the model checker runs in an acceptable amount of time. As we begin to increase the number of concurrent protocol runs, and as we increase the complexity of the model checker itself, we can expect the execution time to increase dramatically. Techniques that increase the efficiency of the model checker are necessary to combat this increase in complexity. In particular, it has become clear that a number of operations can be thought of as independent of each other, in the sense that they can be swapped in the execution trace without affecting the rest of the trace. This leads us to believe that partial order techniques [20] can be applied. The increase in efficiency, ease of use, and expressibility will prove useful in analyzing more complex protocols, including electronic commerce protocols.

References

[1] M. Abadi and A. Gordon. A calculus for cryptographic protocols the spi calculus. In *Proceedings of the Fourth ACM Conference on Computer and Communications Security*, April 1997. To appear.

[2] M. Abadi and M. Tuttle. A semantics for a logic of authentication. In *Proceedings of the 10th ACM Symposium on Principles of Distributed Computing*, pages 201–216, August 1991.

[3] M. Bellare and P. Rogaway. Provably secure session key distribution–the three party case. In *Proceedings of the 27th Annual ACM Symposium on Theory of Computing*, pages 57–66, 1995.

[4] D. Bolignano. An approach to the formal verification of cryptographic protocols. In *Proceedings of the 3rd ACM Conference on Computer and Communication Security*, 1996.

[5] T. Bolognesi and E. Brinksma. Introduction to the iso specification language LOTOS. *Computer Networks and ISDN Systems*, 14(1):25–59, 1987.

[6] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. Technical Report 39, DEC Systems Research Center, February 1989.

[7] D. Craigen and M. Saaltink. Using EVES to analyze authentication protocols. Technical Report TR-96-5508-05, ORA Canada, 1996.

[8] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, March 1989.

[9] H. Garavel. An overview of the Eucalyptus toolbox. In *COST247 workshop*, June 1996.

[10] J. W. Gray and J. McLean. Using temporal logic to specify and verify cryptographic protocols (progress report). In *Proceedings of the 8th IEEE Computer Security Workshop*, 1995.

[11] N. Heintze and J. Tygar. A model for secure protocols and their compositions. *IEEE Transactions on Software Engineering*, 22(1):16–30, January 1996.

[12] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[13] R. Kailar. Accountability in electronic commerce protocols. *IEEE Transactions on Software Engineering*, 22(5), May 1996.

[14] D. Kindred and J. M. Wing. Fast, automatic checking of security protocols. In *USENIX 2nd Workshop on Electronic Commerce*, 1996.

[15] S. Lacroix, J.-M. Boucqueau, J.-J. Quistater, and B. Macq. Providing equitable conditional access by use of trusted third parties. In *European Conference on Multimedia Applications, Services, and Techniques – ECMAST96*, pages 763–782, May 1996.

[16] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer-Verlag, 1996.

[17] C. Meadows. Applying formal methods to the analysis of a key management protocol. *Journal of Computer Security*, 1:5–53, 1992.

[18] C. Meadows. The NRL protocol analyzer: An overview. In *Proceedings of the Second International Conference on the Practical Applications of Prolog*, 1994.

[19] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.

[20] D. Peled. All from one, one for all, on model-checking using representatives. In *Proceedings of the Fifth International Conference on Computer Aided Verification*, Lecture Notes in Computer Science, pages 409–423. Springer-Verlag, 1993.

[21] V. Shoup and A. Rubin. Session key distribution using smart cards. In *Proceedings of Eurocrypt*, 1996.

[22] P. Syverson and P. van Oorschot. On unifying some cryptographic protocol logics. In *Proceedings of the 1994 IEEE Computer Society Symposium on Research in Security and Privacy*. IEEE Computer Society Press, May 1994.

[23] T. Y. C. Woo and S. S. Lam. A semantic model for authentication protocols. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, 1993.

[24] T. Y. C. Woo and S. S. Lam. Verifying authentication protocols: Methodology and example. In *Proceedings of the International Conference on Network Protocols*, 1993.