Using State Space Exploration and a Natural Deduction Style Message Derivation Engine to Verify Security Protocols

E. M. Clarke, S. Jha, and W. Marrero
Carnegie Mellon University
{*emc, sjha, marrero*}*@cs.cmu.edu*

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 adversaries becomes paramount. Many researchers have proposed the use of security protocols to provide these security guarantees. In this paper, we develop a method of verifying these protocols using a special purpose *model checker* which executes an exhaustive state space search of a protocol model. Our tool also includes a natural deduction style derivation engine which models the capabilities of the adversary trying to attack the protocol. Because our models are necessarily abstractions, we cannot *prove* a protocol correct. However, our tool is extremely useful as a debugger. We have used our tool to analyze 14 different authentication protocols, and have found the previously reported attacks for them.

Keywords

Model checking, security protocols, authentication, natural deduction

1 INTRODUCTION

The growth of such entities as the Internet and the World Wide Web have demonstrated the large demand for electronic access to information and for electronic transactions. However, both service providers and consumers need to have some guarantees about reasonable behavior, such as preventing unauthorized access and guaranteeing confidentiality, in the presence of malicious adversaries. Numerous protocols that take advantage of cryptography have been proposed that claim to solve many of the security issues.

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 or from itself. 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 previous 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. One approach has been the use of belief logics to express and deduce security properties [4, 10]. Recently, some researchers have tried to automate the deduction process using theorem provers [5, 11]. Others have provided a rigorous mathematical proof for the correctness of a protocol [2, 26]. Many have tried using formal models to analyze security protocols. Some have developed deductive systems or proof methodologies for their models [1, 3, 6, 7, 9, 20, 25, 27] while others have tried automated search techniques to try to find an error in a model of the protocol [12, 14, 15, 17, 18, 19].

Our approach is also based on model checking and automated search. In this paper we describe a special purpose model checker with two orthogonal components. The first is a *state exploration component*. Each honest agent is described by the sequence of actions that it takes during a run of the protocol, and can be viewed as a finite-state machine. A trace of the actions performed by the asynchronous composition of these state machines corresponds to a possible execution of the protocol by the agents. By performing an exhaustive search of the state space of the composition, we can determine if various security properties are violated.

The second component is the message *derivation engine* which is used to model what the adversary is allowed to do. The derivation engine can be viewed as a simple natural deduction theorem prover for constructing valid messages. We describe the operations that can be performed on messages with a set of inference rules. Because these operations are invertible, each has both an introduction rule and an elimination rule. As in the case of other natural deduction systems, this property guarantees the existence of normalized derivations. For this inference system, the existence of normalized derivations allows for an efficient algorithm for determining whether a message is valid or not.

The standard adversary capabilities found in the literature, which evolved from the Dolev and Yao model [6], fit easily into this framework. The adversary can intercept messages, misdirect messages, and generate new messages using encryption, decryption, concatenation (pairing), and projection. Anytime a message is sent, the adversary intercepts the message and adds it to the set of assumptions it can use to derive new messages. Whenever an honest agent receives a message, the message must have been generated by the derivation engine.

This separation of functionality results in a very intuitive model of computation. In particular, having a black box derivation engine to model the adversary makes the model checker both easier to use and easier to describe and reason about. Unlike term rewriting systems, we do not need to construct a set of rewrite rules to model how an adversary can manipulate participants to generate new messages. In contrast to methods based solely on state space exploration, we need not encode the capabilities of the adversary as a state machine. While a new compiler now provides this feature for the FDR model checker [14], in our system, the capabilities of the adversary are incorporated directly into the model checker itself.

The inclusion of a message derivation engine also means that we do not need to specify ahead of time which messages or which types of messages the model will consider. In theory, the adversary is free to generate any message in an attempt to deceive an honest agent. Moreover, the use of a natural deduction engine allows us to introduce inference rules for new operations, such as XOR and hash functions, easily. Because the derivation engine is an orthogonal component, a new one could be substituted without changing how we model protocols. Finally, we believe this natural deduction framework sheds light on the reasons for the perfect encryption and atomic key assumptions which are frequently made when formal methods are used in this area.

2 INTUITION

In order to concentrate on the security of the protocol itself as opposed to 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 plain-text from the cipher-text.
- There is enough redundancy in the cryptosystem that a cipher-text can only be generated using encryption with the appropriate key. This also implies that there are no encryption collisions. If two cipher-texts are equal, they must have been generated from the same plain-text 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. The drawback is that an attack that takes advantage of a particular property of a specific cryptosystem cannot be found.

We have developed a model checking scheme for the verification of security protocols, and we make use of the same "perfect encryption" assumptions.

We have a very intuitive model which captures the basic idea of message generation and communication. Each role in the protocol, whether the initiator, responder, or server, is described using a sequence of simple commands, such as SEND, RECEIVE, and NEWSECRET, which describe how it interacts with the network during a protocol run.

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 also an unspecified participant which we call the adversary. The adversary models an untrusted communication medium as well as any malicious agents. When messages are sent, they are always intercepted by the adversary, who can then forward them (possibly to someone other than the intended participant). The adversary is also allowed to send messages while impersonating a trusted principal. The adversary may even be selected as a participant in a protocol run.

A *run* of the protocol will then consist of some interleaving of actions from a set of participants (a single session for each role) and from the adversary. A *trace* is the interleaving of one or more runs. We can analyze a trace to determine if the security of the protocol was compromised. In particular we can check if the adversary ever learns a secret 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.

To verify that a protocol is correct, all the possible traces 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 the state of the adversary. Because the length of each run is finite, and we only consider a small number of runs, each trace must necessarily be finite as well. If we can also insure that the number of different traces is finite, then the entire search space will be finite, and we can do an exhaustive search to insure that no reachable state violates the security specification. We will discuss this in more detail in section 5.

3 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 adversary is not allowed to obtain. During the verification, we simply check that the adversary does not have possession of any of the terms in this set. The second property is a temporal property which occurs quite frequently in the literature and which Woo and Lam call *correspondence* [27]. The correspondence relation $X \hookrightarrow Y$ is satisfied if every X event is preceded by a Y event, and there is a one-to-one mapping from X events to Y events.

Many security properties can be expressed as a correspondence relation, and the vast majority of the properties verified in the literature (typically authentication properties) can be expressed as correspondence properties as well. For example, Woo and Lam express authentication by the property that if principal A has finished a protocol run with B, then principal B has at least started a protocol run with A. (Principal B has indeed participated in the protocol) [27]. Mitchell and others check for this property by insuring that if principal A has entered its final state then principal B is no longer in its initial state [19]. Lowe checks that the action *R_running.A.B* (meaning that B is running a protocol in response to A) occurs before the action *I_commit.A.B* (meaning that A has successfully completed a protocol run with B) [13]. Others have checked a weaker property in which the mapping between events need not be one-to-one [12, 24], although these methodologies could check the stronger property as well. There has also been work done on *intensional* specifications which insure that a protocol behaves "as intended," and so necessarily depend on the protocol [23]. It is also easy to see how certain properties of electronic commerce protocols could be expressed this way. For instance, one may want to check that a merchant provides a service only after a client has paid for it and that a client's account is debited only after the merchant has provided the service. Indeed, Leduc and others have verified a kind of electronic commerce protocol by checking for six safety properties which are all expressed as correspondence relations [12].

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.

4 MESSAGES

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

- *Keys* are used to encrypt messages. 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, for all messages m, m_1, and m_2 and keys k, $\{m\}_k \neq m_1 \cdot m_2$, and $\{m\}_k = \{m'\}_{k'} \Rightarrow m = m' \wedge k = k'$.

We also need to consider how new messages can be created from already known messages by encryption, decryption, pairing (concatenation), and projection. The following rules capture this relationship by defining how a message can be derived from some initial set of information I.

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

While this defines the derivability relation $\vdash$, it is not clear if checking $I \vdash m$ is decidable. We will return to this question in section 7.

5 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 which model the honest agents and the adversary. The state of an honest principal is determined by the bindings for its local variables, and by its "program counter." Each run of an honest principal involved in the protocol is modelled as one of these processes and is described by a sequence of actions it is to perform. The initial state of the bindings is assumed to be empty.

The adversary is modelled differently. First, it is not bound to follow any protocol, so it doesn't make sense to describe it as a sequence of actions. At any point in time it is allowed to perform any "realistic" action, which includes intercepting all messages and sending any messages it can generate. The state of the adversary process is completely determined by the set of messages it "knows" and so we model the adversary by keeping track of acquired messages and by using a derivation engine that describes how it can create new messages.

More formally, each honest principal is modelled as a triple $\langle N, p, B\rangle$, 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.
- B: $\mathsf{vars}(N) \rightarrow \mathcal{M}$ is a set of bindings for $\mathsf{vars}(N)$, the set of variables appearing in principal N,

We model the adversary as the pair $\langle Z, I\rangle$, where:

- $Z \in$ *names* is the name of the adversary.
- $I \subseteq \mathcal{M}$ is a set of all messages known by the adversary either as initial information or by eavesdropping. Recall that $\mathcal{M}$ is the set of all possible messages.

The global state is then maintained as the composition of the participating principals, along with the adversary process, a list of secrets, and a set of counters, one for each correspondence relation we are checking. More formally, the global state is a triple $\langle \Pi, C, S\rangle$, where:

- Π is the product of the individual principals and the adversary process. This product is asynchronous, yielding an interleaving semantics, with the restriction that processes synchronize with the adversary on messages.
- C is a set of counters one for each correspondence relation. For each relation $X_i \hookrightarrow Y_i$, there is a counter $C_i \in \mathbb{N}$ whose value is equal to the difference between the number of Y_i events and X_i events that have occurred along the trace so far.
- $S \subseteq \mathcal{M}$ is a set of messages that are are considered secrets. These are the set of words that the adversary is not allowed to know. This set usually includes things like the private keys that principals use to communicate with a server.

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 $\rightarrow$ between principals such that $\mathbf{A} \rightarrow \mathbf{B}$ 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. Communication actions always involve the adversary who controls the network. The action can be thought of as a synchronous transition involving the adversary and the honest principal performing the action. Each receive action corresponds to a message being sent from the adversary to an honest principal and can augment the honest principal's bindings. Each send action corresponds to a message being sent by an honest principal and being received or intercepted by the adversary, thus possibly increasing the set of messages "known" by the adversary. These communication actions are also interleaved with the possible actions of other processes.

The adversary is always willing to receive any messages from any principal, and so send actions are always enabled. When a principal performs a send action, the adversary adds the new message to its local store and the principal takes the following transition:

$$\langle A,\ \textsc{send}(s\text{-}msg).q', B_A\rangle \quad\rightarrow\quad \langle A, q', B_A\rangle$$

In order for a receive action to take place, the message from the adversary must match the message being received. A message $s\text{-}msg \in I$ from the adversary matches a message template $r\text{-}msg$ from principal $\mathbf{B} = \langle B, q, B_B\rangle$, if there exists a substitution $\sigma_B\colon vars(B) \rightarrow \mathcal{M}$ extending B_B ($B_B \subseteq \sigma_B$), such that $s\text{-}msg = \sigma_B(r\text{-}msg)$. If the messages match, then the following transition can be taken:

$$\langle B,\ \textsc{receive}(r\text{-}msg).q', B_B\rangle \quad\rightarrow\quad \langle B, q', \sigma'_B\rangle$$

where σ'_B is the smallest substitution satisfying the conditions above. Because we require that $s\text{-}msg$ match $r\text{-}msg$, if there is already a pair (var, val) in B for some var appearing in $r\text{-}msg$, then the corresponding value in $s\text{-}msg$ must be val. In other words, messages received by honest agents must be consistent with their view of the protocol. Thus the updates to B only add new bindings and never change previous bindings.

For the most part internal actions are used to create new information. For example, NEWSECRET is used to create a nonce or session key, and bind it to a variable. Secrets are globally distinct, and each NEWSECRET action creates a secret that has not appeared up to that point in the protocol. The new binding is added to the principal's set of bindings.

$$\langle A, \textsc{newsecret}(var).p', B\rangle \quad\rightarrow\quad \langle A, p', B'\rangle$$

where $B' = B[var \leftarrow val]$ when val is the new value generated by the action.

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. They have no effect on the state of the principal taking the action. However, the purpose of these actions is to check the correspondence relations, such as $A.\text{ENDINIT}(B) \hookrightarrow B.\text{BEGRESPOND}(A)$ which will maintain a counter in the global state. If this relation is satisfied, then we know that if the principal named A finishes the protocol with B then the principal named B has participated in the protocol with A. We also check that if B finishes, A has participated.

The actions a particular honest principal may make are restricted to the sequence of actions p that represent its role in the protocol. The adversary has no such restriction and is allowed to make any action at any time, provided that it can only send a message *s-msg* to an honest principal if it can derive it ($I \vdash$ *s-msg*). We also place a bound on the number of NEWSECRET actions the adversary can perform, because otherwise the state space would become infinite. In all the protocols we have analyzed, no principal ever checks to see if a key or nonce is new, and so the adversary for these examples never has to take any NEWSECRET actions.

6 SEARCH ALGORITHM

Recall that a trace is an alternating sequence of global states and actions, and that we are interested in checking all possible traces. Clearly, everything in the model is finite except for the set of messages that the adversary can generate. If we can show that the adversary never generates an infinite set of messages, then the entire model is finite and we can perform a depth-first search to check all possible traces of the model, and check that none violate the security properties.

The only times we consider the set of messages the intruder can generate are when we check if it contains a secret, and when we check if the adversary can generate a message that an honest principal is waiting to receive. In the first case, we have a finite set of messages to check, and we prove in section 7 that this is decidable. The later case involves matching against a pattern and in the majority of cases the pattern is restrictive enough to limit the derivation to a finite number of matches (because there are a finite number of atomic messages). In general, however, this need not be the case. In particular, just by using multiple encryption, the set of messages the adversary can generate is infinite. Researchers have used different methods to restrict the model to a finite state space. Lowe restricts the set of messages that the intruder can learn to the set of all atoms along with the set of all encrypted components appearing in protocol messages [14]. Others place type restrictions on variables appearing in messages [8, 12, 19]. Still others solve rewrite equations that can be used to prove that entire families of messages cannot be generated [16,

$$\frac{m_1 \qquad m_2}{m_1 \cdot m_2} \cdot\mathcal{I} \qquad \frac{m_1 \cdot m_2}{m_1} \cdot\mathcal{E}_l \qquad \frac{m_1 \cdot m_2}{m_2} \cdot\mathcal{E}_r$$

$$\frac{m \qquad k}{\{m\}_k} \{\}_k\mathcal{I} \qquad \frac{\{m\}_k \qquad k^{-1}}{m} \{\}_k\mathcal{E}$$

Figure 1 Derivation rules for messages

18]. We plan to augment our matching algorithm to allow for message types; however, we currently limit the state space by placing a bound on the length of the derivations that can be used to find messages that match a variable during the matching process. In this way, we can find "type errors", where, for example, an attack might occur because an honest agent expects to receive a key and accepts a nonce instead. In practice we have been able to get away with a derivation length of 1. Hence, we only generate finite sets of messages, thus the entire state space is finite, and we can use depth first-search to perform an exhaustive search.

7 NORMALIZED DERIVATIONS

In section 4 we discussed how new messages can be generated from known messages using a set of derivation rules. This standard set of adversary capabilities which have evolved from the Dolev and Yao model [6], and which are used by a number of model checkers, fit quite naturally into our framework. In section 5 we discuss when it is necessary to check if the intruder can derive a message. In this section we prove the decidability of this problem by exploiting its similarity to natural deduction and using the idea of normalized derivations. A good discussion of natural deduction and the idea of normalized derivations can be found in [22].

We will assume that the reader is familiar with derivation trees and so we will not formalize them here. We will say that a particular message m is derivable from a set of information I, if there exists a valid derivation tree using the inference rules in Figure 1, such that m appears at the bottom of the tree and all messages appearing at the top of the tree are contained in I. An example of such a tree can be found in Figure 2.

Note that in general there is more than one derivation tree for some message m. While all derivations trees are finite, their sizes are unbounded. In order to prove the decidability of checking $I \vdash m$ we would like to show that there is

$$\dfrac{\dfrac{\dfrac{\{a\}_k \cdot b}{\{a\}_k}\cdot\mathcal{E}_l \qquad k^{-1}}{a}\{\}_k\mathcal{E} \qquad \dfrac{\{a\}_k \cdot b}{b}\cdot\mathcal{E}_r}{a \cdot b}\cdot\mathcal{I}$$

Figure 2 A derivation tree for $\{\{a\}_k \cdot b, k^{-1}\} \vdash a \cdot b$

always a *normalized derivation* of bounded size for which we can search. Unlike normalized derivations in natural deduction, this normalized derivation will have the additional property that all elimination rules ($\cdot\mathcal{E}_l$, $\cdot\mathcal{E}_r$, $\{\}_k\mathcal{E}$) appear above all introduction rules ($\cdot\mathcal{I}$, $\{\}_k\mathcal{I}$). A similar idea for placing bounds on the lengths of derivations can be found in [3, 11, 15, 20]. However, this framework allows for an straightforward translation into a proof search algorithm and explains the reasons behind certain assumptions made about the message space, in particular, the perfect encryption assumption and the atomic key assumption.

Each message construction operation (pairing and encryption) is characterized by a pair of inference rules. One is an introduction rule that creates a new message whose principal connective is that operation. For example the $\{\}_k\mathcal{I}$ rule creates a new encrypted message $\{m\}_k$ from the message m and the key k. The second is an elimination rule that removes that particular operation from the compound message. For example the $\cdot\mathcal{E}_l$ rule takes a message $m_1 \cdot m_2$ and returns its left component m_1. The intuition behind normalized derivations is that an instance of an elimination rule appearing immediately below an instance of the corresponding introduction rule gains no new information. So we transform such a derivation into a smaller derivation in which we eliminate this redundant step.

Using terminology similar to that found in [22], we will call the key k in an instance of the inference rule $\{\}_k\mathcal{E}$ a *minor premise*. Any other premise is a *major premise*. A message that appears in a derivation tree T as the conclusion of an introduction rule and as a major premise of an elimination rule is a *maximum message*. A derivation tree is a *normalized derivation* if it contains no maximum message. We now show that we can transform any derivation tree T into a normalized derivation tree T' by eliminating maximum messages one at a time.

Let T be a derivation tree that is not atomic, and let M be a maximum message in T. Then T', a reduction of T at M is constructed from T using one of the rules below, depending on the form of M. In the diagrams, Π is what would remain of T after removing M and everything above it, while Σ_1, Σ_2, and Σ_3 represent sequences of derivation trees.

$\cdot$-reduction $(i = 1, 2)$

$$\begin{array}{c}\dfrac{\dfrac{\dfrac{\Sigma_1}{m_1} \qquad \dfrac{\Sigma_2}{m_2}}{m_1 \cdot m_2}}{m_i} \\ \Pi \end{array} \qquad \Rightarrow \qquad \begin{array}{c}\dfrac{\Sigma_i}{m_i} \\ \Pi \end{array}$$

$\{\}_k$-reduction

$$\begin{array}{c}\dfrac{\dfrac{\dfrac{\Sigma_1}{m} \qquad \dfrac{\Sigma_2}{k}}{\{m\}_k} \qquad \dfrac{\Sigma_3}{k^{-1}}}{m} \\ \Pi \end{array} \qquad \Rightarrow \qquad \begin{array}{c}\dfrac{\Sigma_1}{m} \\ \Pi \end{array}$$

Theorem 1 *Any derivation tree T for m depending on assumptions A can be transformed into a normalized derivation tree T' for m depending on the same assumptions A.*

Proof: We proceed by induction on the number of maximum messages. If T has no maximum messages then it is already normalized and we are done. Otherwise, take any maximum message M. Because of the perfect encryption assumption, M cannot be the conclusion of an introduction rule for one operator and a major premise in the elimination rule for the other operator. Therefore, one of the reduction rules applies to M. After applying the appropriate reduction rule, we have removed one maximum message and we have not introduced any new ones. The result is a derivation tree for m that depends on the same assumptions and which has one less maximum message. By the induction hypothesis this new derivation tree can be properly transformed.

In fact, the structure of these derivations trees is even more restricted.

Theorem 2 *No introduction rule appears above an elimination rule in a normalized derivation tree.*

Proof: By the definition of normalized, no message appears as the conclusion of an introduction rule and a major premise of an elimination rule. Therefore, we only need consider minor premises. The only minor premises are keys. Recall that we restricted keys to be atomic; therefore, no key can appear as the conclusion of an introduction rule. Hence no message can appear as the conclusion of an introduction rule and a premise of an elimination rule. It follows that no introduction rule appears above an elimination rule.

This theorem provides a simple explanation for the atomic key restriction. As we shall see, this theorem also leads to an efficient algorithm for deciding $I \vdash m$.

8 INFORMATION ALGORITHMS

Theorem 2 suggests an efficient algorithm for determining if $I \vdash m$. Since all elimination rules appear above all introduction rules in a normalized derivation, we can first construct I^*, the closure of the initial set of assumptions I under all elimination rules. We then do a backwards search for a derivation of m from I^* using only introduction rules. (We will use $I^* \vdash_{\mathcal{I}} m$ to denote that such a derivation tree exists.) We will now prove termination and correctness of this algorithm.

Theorem 3 *$I \vdash m$ iff $I^* \vdash_{\mathcal{I}} m$ (Correctness)*

Proof: ($\Rightarrow$) Consider the case when $I \vdash m$. Let T be a normalized derivation tree for m from I. By removing all elimination rules, we get a new tree T' for m from $I \cup \Delta$, where Δ is the set of all the messages appearing at the top of T' that are not in I. So T' is a derivation tree for $I \cup \Delta \vdash_{\mathcal{I}} m$. By construction, each $\delta_i \in \Delta$ can be derived from I using only elimination rules so $I \cup \Delta \subseteq I^*$. Therefore T' is also a derivation tree for $I^* \vdash_{\mathcal{I}} m$.

($\Leftarrow$) Consider the case when $I^* \vdash_{\mathcal{I}} m$. By definition, $I \vdash i$ for each $i \in I^*$. Therefore, we can transform the tree T' for $I^* \vdash_{\mathcal{I}} m$ into a tree T for $I \vdash m$ by placing a derivation tree T_i for $I \vdash i$ above each message $i \notin I$ at the top of T'.

Theorem 3 proves the correctness of our algorithm. To prove termination, consider all the messages that are generated during the derivation. The elimination rules start from the assumptions and generate only sub-messages. Since the original messages are finite length, and since there are only a finite number of them, the process of closing under elimination rules terminates. Now consider the backwards search using introduction rules. Since these rules are applied backwards, the new major premise messages we search for must be sub-messages, and so that part of the search terminates when we reach a message in I^* or we reach an atomic message that is not in I^*. The minor premises we search for are all atomic keys, and so clearly, that involves a simple scan of I^* as well. Therefore, the entire algorithm terminates.

The implementation of this algorithm is given in the following two figures. In Figure 3 we consider how to update the adversary's set of information when

```
funct add(I, m)
  foreach i ∈ I do
    if i = {x}_y ∧ m = y^{-1}
      then I := add(I, x)
           if y ∈ I then I := I − i fi
    fi
  od
  if m ∈ A
    then return I ∪ {m}
  elsif m = x · y
       then return add(add(I, x), y)
  elsif m = {x}_y ∧ y^{-1} ∈ I
       then if y ∈ I
                then return add(I, x)
                 else return add(I ∪ {m}, x)
            fi
  else
       return I ∪ {m}
  fi
```

Figure 3 Augmenting the adversary's knowledge

it learns a new message. Anytime the adversary gains a new message, we add it to the set of messages it currently has and we close this set under elimination rules while also removing "redundant" messages. Figure 4, describes how to search for a derivation of m from I^* using only introduction rules. This search involves first checking if $m \in I^*$. If this fails, then we recursively search for the components of m.

```
funct in(I, m)
  if m ∈ I
    then return true
  elsif m = x · y
       then return in(I, x) ∧ in(I, y)
  elsif m = {x}_y
       then return in(I, x) ∧ in(I, y)
  else
       return false
  fi
```

Figure 4 Searching the adversary's knowledge

9 WOO AND LAM EXAMPLE

We now analyze a protocol published by Woo and Lam in [28]. The protocol is given below:

1. $A \rightarrow B : A$
2. $B \rightarrow A : N_b$
3. $A \rightarrow B : \{N_b\}_{K_{as}}$
4. $B \rightarrow S : \{A \cdot \{N_b\}_{K_{as}}\}_{K_{bs}}$
5. $S \rightarrow B : \{A \cdot N_b\}_{K_{bs}}$

Here A is the initiator, B is the responder, and S is a server. In message one, A initiates the protocol by sending B its name. B replies in message two with a nonce. In message three, A encrypts the nonce with the secret key it shares with the server and sends it back to B. B then concatenates this with A's name, encrypts the whole thing with its private key with the server and sends it to the server in message four. In the last message, the server can decrypt using both keys to recover the original nonce and return it to B along with A's name and encrypted with B's key.

In order to use our model checker, we first isolate which actions are performed by A, which actions are performed by B, and which are performed by S. We then write a short sequence of actions which make up each participant's role in the protocol. The process description for principal B can be found in Figure 5. The descriptions for the other honest agents are similar. All that remains is to specify the initial state of the adversary's local store. Initially, the adversary knows the names of all agents and its own secret key with the server.

In just a few seconds, our model checker finds the following attack:

1. $I(A) \rightarrow B : A$
2. $B \rightarrow I(A) : N_b$
3. $I(A) \rightarrow B : N_b$
4. $B \rightarrow I(S) : \{A \cdot N_b\}_{K_{bs}}$
5. $I(S) \rightarrow B : \{A \cdot N_b\}_{K_{bs}}$

In this case B is the only honest agent participating in the protocol. The adversary pretends to be both the server S and the other honest agent A. In the end, B believes that A has participated in a run of the protocol, while A has not participated at all. The error occurs in step 3 of the protocol. B is waiting for a message that is encrypted with a key it does not possess; therefore, B has no idea what the message should look like. The adversary can replay the message B sent in step 2 and B will accept it as coming from A in step 3. B then generates message 4 by concatenating this message with A's name and encrypting with the secret key it shares with the server. The problem now is that this message just sent by B in step 4, is the exact message B is expecting from the server in step 5. So again, the adversary just replays this message to B in order to complete the attack.

This example demonstrates a type of attack that depends on the fact that an agent cannot differentiate between one kind of message and another. In this example, the responder does not possess the key K_{as} and so cannot tell in step 3 whether the message it receives is encrypted or not. Actually, the situation is worse, since we assume that B knows nothing about the message it receives and could actually accept *any* message at this point. However, not all messages generate an attack. By accepting its own nonce in step 3, B generates and sends in step 4 the exact message it is expecting in step 5. It is this interplay that allows the attack.

This kind of attack illustrates one of the trade-offs involved when trying to make the message space finite. When we look at the model of the responder in this protocol in Figure 5, we notice that the second receive performed by the responder consists of a single message variable. This is because, as discussed above, from the responder's point of view, this message could be anything. If we place a type restriction on this variable, so that it can only unify against an encrypted nonce, the attack discussed here would not be found. However, this variable matches an infinite number of messages derivable by the adversary and so we must somehow limit the set of messages we consider. In this case, the attack is found even if we limit the size of the derivations that the adversary is allowed to perform when generating messages to a single step (i.e. to messages directly contained in I^*).

```
((begrespond (*var* a))
 (receive (*var* a)
          (*var* a))
 (newsecret (*var* nb))
 (send (*var* a)
       (*var* nb))
 (receive (*var* a)
  (*var* tos))
 (send s
       (encrypt (*secret* b)
       (concat (*var* a) (*var* tos))))
 (receive s
  (encrypt (*secret* b)
   (concat (*var* a) (*var* nb))))
 (endrespond (*var* a)))))
```

Figure 5 Process description for the responder

10 CONCLUSION

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, with a single session for each role, 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 the 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 the 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 [21] can be applied. The increase in efficiency, ease of use, and expressibility will prove useful in analyzing more complex protocols, including electronic commerce protocols.

Finally, there is the problem of how many runs to check. Because the size of the model grows exponentially with the number of runs, it would be ideal to be able to limit the search to only one or two runs. Inspired by the kind of analysis done by Lowe [13], we are investigating a framework in which one can analyze and prove bounds on the number of runs that need to be checked. We use a kind of dependence graph on the messages that the adversary needs to generate. For some simple protocols, we have been able to argue that if there is a flaw in a protocol, then that flaw will be discovered by a model checker on a model with only one run. We plan to continue investigating this methodology with the hope of being able to automate it as well.

ACKNOWLEDGEMENTS

The authors would like to thank Natarajan Shankar and Carsten Schürmann for helpful discussions regarding normalized derivations and for feedback on an earlier version of this paper.

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. 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.

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

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

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

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

[7] 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.

[8] N. Heintze, D. Tygar, J. Wing, and H. Wong. Model checking electronic commerce protocols. In *Proceedings of the USENIX 1996 Workshop on Electronic Commerce*, pages 146–164, 1996.

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

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

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

[12] G. Leduc, O. Bonaventure, E. Koerner, L. Léonard, C. Pecheur, and D. Zanetti. Specification and verification of a TTP protocol for the conditional access to services. In *Proceedings of the 12th J. Cartier Workshop on Formal Methods and their Applications: Telecommunications, VLSI and Real-Time Computerized Control System*, October 1996.

[13] 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.

[14] G. Lowe. Casper: A compiler for the analysis of security protocols. In *Proceedings of the 1997 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 18–30, 1997.

[15] W. Marrero, E. Clarke, and S. Jha. Model checking for security protocols. Technical Report CMU-CS-97-139, Carnegie Mellon University, 1997.

[16] C. Meadows. A model of computation for the NRL protocol analyzer. In *Proceedings of the 1994 Computer Security Foundations Workshop*. IEEE Computer Society Press, June 1994.

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

[18] J. Millen. The Interrogator model. In *Proceedings of the 1995 IEEE Symposium on Security and Privacy*, pages 251–260. IEEE Computer Society Press, 1995.

[19] J. C. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using murϕ. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 1997.

[20] L. Paulson. Proving properties of security protocols by induction. In *Proceedings of the 1997 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 70–83, 1997.

[21] 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.

[22] D. Prawitz. *Natural Deduction: A Proof-Theoretical Study.* Almqvist & Wiksell, 1965.

[23] A. W. Roscoe. Intensional specifications of security protocols. In *9th Computer Security Foundations Workshop*, 1996.

[24] S. Schneider. Security properties and CSP. In *Proceedings of the 1996 IEEE Computer Society Symposium on Research in Security and Privacy*, 1996.

[25] S. Schneider. Verifying authentication protocols with CSP. In *Proceedings of the 1997 IEEE Computer Society Symposium on Research in Security and Privacy*, 1997.

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

[27] 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.

[28] T. Y. C. Woo and S. S. Lam. A lesson on authentication protocol design. In *Operating Systems Review*, pages 24–37, 1994.

Edmund M. Clarke received a B.A. degree in mathematics from the University of Virginia in 1967, an M.A. degree in mathematics from Duke University in 1968, and a Ph.D. degree in Computer Science from Cornell University in 1976. After receiving his Ph.D., he taught in the Department of Computer Science, Duke University, for two years. In 1978 he moved to Harvard University where he was an Assistant Professor of Computer Science in the Division of Applied Sciences. He left Harvard in 1982 to join the faculty in the Computer Science Department at Carnegie Mellon University. He was appointed Full Professor in 1989. In 1995 he became the first recipient of the FORE Systems Professorship, an endowed chair in the School of Computer Science.

Dr. Clarke's interests include software and hardware verification and automatic theorem proving. In his Ph.D. thesis he proved that certain programming language control structures did not have good Hoare style proof systems. In 1981 he and his Ph.D. student Allen Emerson first proposed the use of *Model Checking* as a verification technique for finite state concurrent systems. His research group pioneered the use of Model Checking for hardware verification. Symbolic Model Checking using BDDs was also developed by his group. This important technique was the subject of Kenneth McMillan's Ph.D. thesis, which received an ACM Doctoral Dissertation Award. In addition, his resarch group developed the first parallel resolution theorem prover (Parthenon) and the first theorem prover to be based on a symbolic computation system (Analytica).

Dr. Clarke has served on the editorial boards of Distributed Computing and Logic and Computation and is currently an editor-in-chief of Formal Methods in Systems Design. He is on the steering committees of two international conferences, Logic in Computer Science and Computer-Aided Verification. He is a Fellow of the Association for Computing Machinery, a member of the IEEE Computer Society, Sigma Xi, and Phi Beta Kappa.

Somesh Jha is a post-doctoral fellow in the Robotics Institute at Carnegie Mellon University. Somesh Jha received his B. Tech in Electrical Engineering from IIT Delhi, India, Masters in Computer Science from Penn State, and Ph.D. in Computer Science from Carnegie Mellon University. Somesh has also worked as a computer consultant for four years. He has worked as a consultant for IBM, AT&T, and UPS. His main areas of interests are in formal methods with applications to communication protocols, software engineering, and multi-agent systems. He is also interested in computational finance and application of economics to software engineering decision making.

Will Marrero is a graduate student in the School of Computer Science at Carnegie Mellon University. Will Marrero received his B.A. in mathematics and computer science from Columbia University in 1992. He has worked as a summer intern doing formal verification at Intel and at Cadence. His main areas of interest are formal methods as applied to security protocols, communication protocols, and hardware design.