Applying Formal Methods to the Analysis of a Key Management Protocol

Catherine Meadows
Code 5543
Center for Secure Information Technology
Naval Research Laboratory
Washington, DC 20375

ABSTRACT

In this paper we develop methods for analyzing key management and authentication protocols using techniques developed for the solutions of equations in a term rewriting system. In particular, we describe a model of a class of protocols and possible attacks on those protocols as term rewriting systems, and we also describe a software tool based on a narrowing algorithm that can be used in the analysis of such protocols. We formally model a protocol and describe the results of using these techniques to analyze security properties. We show how a security flaw was found, and we also describe the verification of a corrected scheme using these techniques.

1 Introduction

It is difficult to be certain whether or not a cryptographic protocol satisfies its requirements. In a number of cases subtle security flaws have been found in protocols some time after they were published. These flaws were independent of the strengths or weakness of the cryptographic algorithms used. Examples include the Needham-Schroeder key distribution protocol [23] which was found by Denning and Sacco [7] and Bauer, Berson and Feiertag [1], to be vulnerable to various kinds of replay attacks that can be used whenever old keys are compromised, the software protection scheme of Purdy, Simmons, and Studier [25], for which Simmons [28] showed how a penetrator could combine previously generated messages in such a way that the system could be induced to grant unauthorized access to software, and a protocol in the CCITT X.509 draft standard [5], which was shown by Burrows, Abadi, and Needham [4] to be vulnerable to replay attacks even when keys are not compromised. Other similarly flawed protocols are discussed by Moore in [21].

As more systems are designed that depend upon such protocols for more varied and complex security needs, it is likely that other flaws will arise. If systematic means of assuring correctness are not used, these flaws may not be discovered until significant damage has been done.

One approach to the problem of assuring correctness that has been suggested, for example by Kemmerer in [13], is to use machine aided formal verification techniques. The protocol and its desirable security properties are modeled in a formal specification language, and a machine verification system is used in an attempt to prove that these properties hold. If the attempt succeeds, one has gained greater assurance that the protocol satisfies its requirements. If it fails, then the examination of the reasons for the failure may point out security flaws in the protocol.

Techniques like these have been applied in the design of communication protocols and of secure computing systems. Thus it seems likely that they will be useful in the analysis of key management protocols, which have some of the properties of both. Yet so far they have found little application. Part of the reason for this may be that existing machine verification systems do not emphasize the theorem-proving techniques that would be most useful in the analysis of such protocols. In this paper we attempt to fill this gap by showing how a formal model and a software tool implementing specialized theorem-proving techniques, originally described by the author of this paper in [17], were successfully applied to the analysis of the selective broadcast protocol designed by G. J. Simmons [28]. The fact that the application of these techniques uncovered a subtle security flaw in this protocol shows that they promise to be useful.

The rest of the paper is organized as follows. In Section 2 we compare our approach with other work on the specification and verification of cryptographic protocols and give a brief survey of other work in the field. In Section 3 we describe the selective broadcast protocol and the results of our analysis. In Section 4 we describe our formal model, which is an adaptation of Dolev and Yao's [8] term-rewriting system model of public-key protocols. We also describe the proof techniques we use and a Prolog program that assists us in applying them. In Section 5 we describe the formal specification of the protocol. In Section 6 we give an outline of our analysis and show how it uncovered a security flaw. In Section 7 we discuss the implications of our results and ways in which our system could be improved.

2 Comparison With Other Work

Approaches taken in the formal verification and analysis of security properties of cryptographic protocols that are independent of the strengths or weaknesses of the cryptoalgorithms used fall into four main categories. The first of these is the attempt to model and verify the protocol using specification languages and verification tools not specifically developed for the analysis of cryptographic protocols. This, for example, is the approach taken by Britton in [3], by Kemmerer in [13], and by Viradharajan in [33].

Another is to develop expert systems which a protocol designer can use to try out different scenarios. This is the approach followed by Millen in [20], Longley and Rigby in [15], and to a limited extent by Kemmerer in [13]. That this approach can be successfully used to detect previously undiscovered flaws in a protocol has been shown by Longley and Rigby in [15]. However, these systems do not guarantee that the investigation of a protocol's security is complete; thus, although they may be useful in identifying flaws, they can not be used to provide greater assurance of a protocol's security.

A third approach is to model the requirements of a protocol formally using logics developed for the analysis of knowledge and belief. This, for example, is the approach taken by the Burrows, Abadi, and Needham logic [4] and its extensions (for example [10], by Moser in [22], and by Rangan in [26]. Formal semantics for such logics have been developed by Syverson [30, 31] and Bieber [2]. This approach has resulted in the discovery of security flaws in protocols as is reported in [4]. But the result of analysis using such logics is not precisely to get a proof of security of a protocol that tells the analyst whether or not a certain undesirable event can occur. Rather one gets an analysis of the kinds of trusts and beliefs that are necessary for a cryptographic protocol to complete successfully. (See for example Nessett's counterexample [24] and the analysis by Syverson in [32].) Thus one can think of these methods as complementary to the kinds of techniques that we have presented in this paper.

The fourth approach is to develop a formal model based on the algebraic term-rewriting properties of cryptographic systems. This is the approach followed by Dolev and Yao's public-key model [8], upon which our model is based, and the model of Kasami, Yanamura, and Mori [12], also used by Lu and Sundareshan in the verification of a hierarchical key management protocol [16].

Although our model is based on the Dolev-Yao model, it is more similar to the Kasami-Yanamura-Mori model in that Dolev and Yao concentrate on protocols that can be completely modeled as term-rewriting systems and for which efficient algorithms for deciding security problems can be found, while we and Kasami et al. follow a more general approach. The main differences between our approach and the Kasami-Yanamura-Mori and Dolev-Yao approaches is that we provide automated support for the security analysis and we allow a user to query the system about the attainability of internal system states as well as the obtainability of secret words by a penetrator. This allows us to model security goals such as authentication in a more natural way.

3 Description of the Selective Broadcast Protocol

3.1 The Protocol

The selective broadcast protocol described in [28] is intended to provide a means by which one participant can encrypt a message so that it can only be decrypted by a designated subset of the other participants, and so that participants with the ability to decrypt a message cannot pass on the ability to other participants. The protocol relies for its security on a combination of encryption and tamperproof processors that will only perform certain actions prescribed by the protocol. Messages may either be released to the participants when decrypted, or stored in the processors and not made directly available to the participants. Examples of applications of this protocol in the latter case are software protection, in which the messages would be software that could only be executed inside the tamper-proof processors, and key distribution, in which the messages would be session keys that could never appear in the clear outside the processors. In our specification and analysis of the protocol, we assumed that the latter case held, since this was the assumption under which the original software protection protocol held.

Since the protocol relies for its security on the correct operation of the processors, the set of decisions that must be made by the processors is made as simple as possible, and the amount of data that must be stored is kept at a minimum. This means that the processors do not do much checking of input data, and thus will perform their operations on just about any data that is input. Hence it must be shown that there exists no set of data obtainable by a penetrator that, when input, could result in a violation of the protocol's security requirements, even when the penetrator is a legitimate user or set of users of the system who may have access to one or more processors. Indeed, it was a flaw of this sort in an earlier, similar, protocol [25], that lead to the development of the selective broadcast protocol.

We use the notation e(X,Y) to mean the encryption of word Y with key X, and d(X,Y) to mean the decryption of word Y with key X. Multiple encryption is represented by the use of nested functions; e.g., if we encrypt word Z with key Y, and then decrypt that using the decryption of word X with word W, that is denoted by d(d(W,X),e(Y,Z)).

The protocol relies upon single-key cryptography, but no other assumption is made about the cryptoalgorithm used. The protocol runs as follows. Each participant has access to a tamper-proof processor A which has access to a unique secret key q(A). The processors also have access to a common key k. Both k and q(A) are only available to the processors, and never appear in the clear outside the processors. Messages are only decrypted inside the tamper-proof processors. Each processor has a unique identifier e(k,q(A)), which is made public. If participant A wishes participant B to be able to use message M, he first chooses an identifier J for M and has his processor encrypt M using the key d(q(A),d(k,J)). He then sends the encrypted message and an encrypted key e(k,e(q(B),d(q(A),e(q(B),d(q(A),d(k,J)))))) to B. B inserts key and message, along with the identifier e(k,q(A)), into his processor. The processor uses its knowledge of k, e(k,q(A)), and q(B) to decrypt the key, uses the key to decrypt the message, and then uses the message according to the rules of the application.

Each processor can be run in three modes: encrypt message, encrypt key, and decrypt message. In each mode, the user inserts two K-bit words X and Y (and possibly a third word M) and the processor outputs e(k,q(A)), where q(A) is the processor's identifier, and e(k,e(d(k,X),d(q(A),e(d(k,X),d(q(A),d(k,Y)))))).

In the encrypt message mode, the user inserts X = e(k,q(A)), Y = J, and M = message, where J is the identifier of the message. The second output of the processor reduces to J. This is used to check that the system is working properly. The processor also outputs e(d(q(A),e(d(k,X),d(q(A),d(k,Y)))),M), which reduces to e(d(q(A),d(k,J)),M), the encrypted message.

In the encrypt key mode, the user inserts X = e(k,q(B)) and Y = J, where B is the processor to which he wishes to send the message. In this case the second output of the processor reduces to e(k,e(q(B),d(q(A),e(q(B),d(q(A),d(k,J)))))), which is the encrypted key.

In the decrypt message mode, the owner of processor B inputs

X = e(k,q(A));
Y = e(k,e(q(B),d(q(A),e(q(B),d(q(A),d(k,J)))))), and;
M = e(d(q(A),d(k,J)),message).

In this case the second output reduces to J. This is used to check that the key transfer has taken place correctly, and to provide authentication of the message. The processor also decrypts the key and uses it to decrypt the message, that is, to compute d(d(q(B),e(d(k,X),d(q(B),d(k,Y)))),M), which produces the decrypted message inside the processor. The processor then uses the decrypted message according to the rules of the application.

This protocol is relatively simple. We were able to specify it using only eleven protocol rules and three rewrite rules, thus making an analysis that proceeded largely by hand practical. However, the complexity of the operations used, which made it difficult for an analyst to gain an intuitive feeling for the security of the protocol, and its reliance on mechanisms similar to those employed by an earlier, flawed, protocol [25], suggested that a formal analysis was necessary. These facts made the protocol an excellent test case for our methods.

3.2 The Attack

Our analysis found a means by which a penetrator could induce a participant to believe that he was decrypting a message supplied by some other honest participant when he was actually decrypting a message supplied by the penetrator. By supplying incorrect information to his processor when it is run in encrypt message mode, a penetrator can encrypt his own message using another processor's key. This allows a penetrator to impersonate other members of the network. This flaw can be corrected by relying on the processor, instead of the protocol participant, to supply the first word input when it is run in encrypt message mode. We altered the specification to correct this flaw, and used our techniques to prove that the altered specification satisfied the desired security property.

The attack on the original protocol works as follows. We assume that the penetrator has access to some module x. First, the penetrator purchases the right to program p with identifier j from some honest user a. User a sends

e(k,q(a))
e(k,e(q(x),d(q(a),e(q(x),d(q(a),d(k,j))))))
e(d(q(a),d(k,j)),p)

to the penetrator. The first word is a's encrypted identifier, the second is the encrypted key, and the third is the encrypted message. The penetrator submits the following words to his module in encrypt message mode:

X = e(k,q(a))
Y = e(k,e(q(x),d(q(a),e(q(x),d(q(a),d(k,j))))))
Z = px

where px is a program supplied by the penetrator, and the second word supplied is the encrypted key supplied by a. The third word output by the module is

e(d(q(x),e(d(k,e(k,q(a))),d(q(x),d(k,e(k,q(x),d(q(a),e(q(x),d(q(a),d(k,j))))))))),px)

which reduces to

e(d(q(a),d(k,j)),px).

Now suppose that user b requests program p with identifier j from user a. User a sends

e(k,q(a))
e(k,e(q(b),d(q(a),e(q(b),d(q(a),d(k,j))))))
e(d(q(a),d(k,j)),p)

to b. The penetrator intercepts the message and replaces the third word by e(d(q(a),d(k,j)),px). He then sends the message to b. User b inserts the three words in his module in decrypt message mode. The module outputs the words

e(k,q(b))
e(k,e(d(k,e(k,q(a))),d(q(b),e(d(k,e(k,q(a))),d(q(b),e(q(b),d(q(a),e(q(b),d(q(a),d(k,j))))))))))

which reduces to j. Thus the module believes that it has received the correct message, and produces the decrypted message

d(d(q(b),e(d(k,e(k,q(a))),d(q(b),d(k,e(k,e(q(b),d(q(a),e(q(b),d(q(a),d(k,j)))))))))),e(d(q(a),d(k,j)),px))

which reduces to px.

3.3 The Fix

In our analysis of the protocol, we were able to prove that a penetrator could successfully pass off his message px as user a's if and only if he was able to produce the word e(d(q(a),d(k,j)),px). Thus, in order to modify the protocol so that it would be secure, all that was necessary to do was to make it impossible for the penetrator to produce that word. We did this by requiring that the module itself supply the first word input when it is run in encrypt message mode. This can be done since that word is just the encrypted identifier of the module. We then proved a general theorem which implied that any state that was unreachable in the old protocol was also unreachable in the new one. This meant that we did not have to reverify old security results, which were defined in terms of unreachability of states. Finally, we showed that the word e(d(q(a),d(k,j)),px) was unobtainable in the new version of the protocol.

In the remainder of this paper, we will show how we modeled the protocol, and how we discovered the flaw and verified the security of the corrected protocol.

4 Description of the Model and Proof Techniques Used

The model we use is an adaptation of the public-key model developed by Dolev and Yao in [8]. We consider a protocol as a set of rules for passing messages between the participants. A participant in the protocol who receives a message will, if he accepts it as genuine, generate a new message by performing certain operations upon it or other messages received earlier. Thus a cryptographic protocol may be thought of in part as a set of rules for generating words in some formal language. In symbolic terms, we can think of these operations as being applied in two steps: first operations are applied to a word or set of words, and then algebraic properties of the operations are used (such as the fact that encryption cancels out decryption with the same key and vice versa) to produce the actual words generated. Since in many cases the algebraic properties of the operations involved can be interpreted as reduction rules (that is, a set of rules for transforming words into words that are "simpler" according to some well-defined measure), the protocol may be thought of in part as a set of rules for generating words in a term-rewriting language. A penetrator who tries to break the protocol by intercepting messages, supplying false messages to the participants, and performing operations on messages himself in order to find out a secret word, may be thought of as attempting to determine whether a particular word belongs to a given term-rewriting language.

In Dolev and Yao, the security of a protocol is expressed in terms of a penetrator's being unable to generate words belonging to some predetermined set. Thus, for example, a protocol may be considered secure if a penetrator is unable to discover secret data not intended for him. In our model this notion of security is extended to cover the penetrator's inability to cause internal state variables to take on certain values. Thus we might say that a protocol is insecure if a penetrator is able both to learn a word K and convince an honest participant that K is a session key good for communication with some other honest participant.

In our model protocol rules are expressed as statements of the form

If $I \subseteq \mathbf{W}$ and A and Cond(I,A) then $\mathbf{W} := \mathbf{W} \cup O$ and A'

where $\mathbf{W}$ denotes the set of words known by the penetrator, I denotes the words making up a message sent to a participant in the protocol, O denotes the words making up a message sent by a participant in the protocol, A and A' are statements consisting of conjunctions of equations of the form $S = T$, where S is a state variable name, and Cond(I,A) is a set of further conditions on I and A of the form $C \neq D$, where C is a subterm of I or A, and D is a term. Rules may be either deterministic or nondeterministic, in the sense that a rule may either always apply, or only apply under circumstances which have not yet been specified. Rules of this form may be used to describe events as various as:

1. A participant in the protocol responding to a message from the penetrator;
2. A change in the internal state values of a participant;
3. A participant initiating an instance of a protocol, or;
4. The penetrator using the facilities available to him to generate a message.

For example, if the penetrator is able to encrypt data, we can represent this by a rule

$$\boxed{\text{If } \{X,Y\} \subseteq \mathbf{W} \text{ then } \mathbf{W} := \mathbf{W} \cup \{e(X,Y)\}}$$

where e(X,Y) denotes the result of encrypting message Y with key X. On the other hand, if a participant in a protocol will encrypt all messages sent to it with the current session key and send them out, this may be described by the rule

$$\boxed{\text{If } \{Y\} \subseteq \mathbf{W} \text{ and } \mathit{KEYSTATE}(a) = X \text{ then } \mathbf{W} := \mathbf{W} \cup \{e(X,Y)\}.}$$

The specification also includes a set of *rewrite rules* (also known as *reduction rules*) that describe ways in which complex words may be reduced to simpler ones. Examples of such rewrite rules are:

1. d(X,e(X,Y)) $\rightarrow$ Y where e(A,B) denotes encryption of word B with key A, and d(A,B) denotes decryption of word B with key A, and;
2. p(s(X)) $\rightarrow$ X where s and p are the successor and predecessor operators, respectively.

Thus we make the following definition of a specification of a cryptographic protocol.

Definition 3.1: A *specification* **S** of a protocol is a tuple

$[T(\Phi,F), \hat{E}, \mathbf{S}, \mathbf{R}, W_0, A_0]$

where Φ is a countable set of variables, F is a finite set of function symbols, $T(\Phi,F)$ is a set of terms made up from Φ and F, $\hat{E}$ is a set of rewrite rules defined on $T(\Phi,F)$, **S** is a set of state variable names, , W_0 is set of words from $T(\Phi,F)$ initially known to the penetrator, A_0 is a set of initial assignments of values to the state variables, and **R** is a set of protocol rules of the form

$$\boxed{\text{If } I \subseteq \mathbf{W} \text{ and } A \text{ and Cond}(I,A) \text{ then } \mathbf{W} := \mathbf{W} \cup O \text{ and } A'}$$

where I and O consist of terms from $T(\Phi,F)$, Cond(I,A) is a set of conditions on I and A, and A and A' are conjunctions of equations of the form $T = D$, where T is a state variable name and $D \in T(\Phi,F)$. As before, I denotes the words input into the rule, A denotes the state variable values input, O denotes the words output, and A' denotes the state variable values output.

Definitions of terms and related ideas are given below. The concepts are standard, but the particular definitions used are from [34].

Definition 3.2: Let Φ be a countable set of variables and F a family of function symbols disjoint from Φ with associated arity. A *term* is either a variable or a function symbol followed by n terms, where n is the arity of the function symbol. A function symbol of arity zero is called a *constant*. Let $T(\Phi,F)$ denote the set of all terms made up from Φ and F. A *substitution* is a function from Φ to $T(\Phi,F)$ that is the identity on all but a finite subset of Φ. The term obtained by applying a substitution σ to the variables of a term F is denoted by σT. A *unifier* of two terms F and G is a substitution σ such that σF = σG. A *most general unifier* of F and G is a unifier σ of F and G, such that, if τ is another unifier, there is another unifier μ such that $\tau = \mu\sigma$. Most general unifiers are unique up to renaming of variables.

Since our analyzer is written in Prolog, we will follow its conventions and represent function symbols by lower-case words and variables from Φ by capital letters or words beginning with capital letters. Variables not from Φ, such as, for example, the variable I used to represent the words input into a rule, will be represented by italicized capital letters or italicized words beginning with capital letters.

Definition 3.3: A *term-rewriting system* over $T(\Phi,F)$ is a set of directed equations $\hat{E}$ such that, for $T_1 \rightarrow T_2$ in $\hat{E}$, V(T_2) $\subseteq$ V(T_1), where V(T) denotes the set of variables appearing in T. $\rightarrow_{\hat{E}}$ is the finest relation over $T(\Phi,F)$ containing $\hat{E}$ and closed by substitution of terms for variables and replacement of subterms by their reduced forms. If A is a term, and there is a unique irreducible term that can be obtained by applying reductions to A, we call that term nf(A), or the *normal form* of A. A substitution $\sigma T(\Phi,F)$ is said to be a *unifier of U and T with respect to $\hat{E}$* if σU and σT are reducible to the same term using repeated applications of the rewrite rules in $\rightarrow_{\hat{E}}$. A set of unifiers Σ of A and $B$with respect to $\hat{E}$ is *complete* if, for any unifer τ of A and B with respect to $\hat{E}$, there is a substitution μ and a $\sigma \in \Sigma$ such that $\tau = \mu\sigma$.

We will be interested in term-rewriting systems that are *Noetherian* and *locally confluent*. These are defined below.

Definition 3.4: Let $\hat{E}$ be a term-rewriting system over $T(\Phi,F)$. We say that $\hat{E}$ is *Noetherian* if, whenever term A is reachable from term B via a sequence of applications of reduction rules, then term A is reachable from term B via a finite sequence of such applications. We say that $\hat{E}$ is *locally confluent* if whenever A is reducible to B and to C by two different single reductions, then B and C are both reducible to a fourth term D.

Noetherian, locally confluent systems have the property that every term A is reducible to a unique irreducible normal form nf(A). See Huet and Oppen [11] for a discussion.

We now define what we mean by a system state, and show how we describe states.

Definition 3.5: A *system state* is a pair (Λ,Δ) where Λ gives the set of words known by the penetrator and $\Delta = \gamma_1 = \beta_1,...,\gamma_k = \beta_k$ gives the values of the internal state variables γ_1, ..., γ_k. A *state description* (L,D) is a pair such that L is a set of terms from $T(\Phi,F)$ and $D = G_1 = D_1,...,G_t = D_t$ is the assignment of a set of terms D_i from $T(\Phi$,S) to state variables G_i. We say that (Λ,Δ) *satisfies* the description (L,D) if all terms in (Λ,δ) are irreducible and there is a substitution σ such that σL $\subseteq \Lambda$ and σD $\subseteq \Delta$.

Thus, for example, given the state description ({e(k,X)},{}), we assume that the penetrator knows a word e(k,σX), where σ is a substitution assigning X to a term such that e(k,σX) remains irreducible. Hence the penetrator might know e(k,m) or e(k,e(r,s)), but not e(k,d(k,m)), which reduces to m.

The assumption of irreducibility is necessary in order for the definition of state description to be meaningful. For example, suppose that we did not enforce this restriction, and we assumed that the penetrator knew some word of the form e(k,X). Then, since X could equal d(k,Y), and e(k,d(k,Y)) reduces to Y, all this would mean would be that the penetrator knows some word Y. This is not very useful information.

A rule may now be interpreted as a means of moving from state description to state description as follows. The conditions of a rule set out a state description that must be satisfied for a rule to apply, and the conclusions give the rules for constructing the new state description that holds after the rule has fired. This notion is set out formally below.

Definition 3.6: Let (L,D) and (L',D') be state descriptions. Let R be a rule of the form

$\boxed{\text{If } I \subseteq \mathbf{W} \text{ and } A \text{ and Cond}(I,A) \text{ then } \mathbf{W} := \mathbf{W} \cup O \text{ and } A'}$

such that there is a substitution σ assigning terms to the variables of R and leaving the terms of (L,D), (L',D'), I, and A irreducible, such that

a) $\sigma I \subseteq \sigma L$;

b) $\sigma A \subseteq \sigma D$;

c) Cond(σA,σI) is not contradicted (that is, Cond(σA,σI) contains no inequality of the form $X \neq X$);

d) $\sigma L' \subseteq \sigma L \cup \text{nf}(\sigma O)$;

e) $\sigma D' \subseteq (\sigma D - \sigma D'') \cup \text{nf}(\sigma A')$, where $\sigma D''$ is the set of assignments in σD that are replaced by new assignments in $\sigma D'$ or nf(σA), and;

f) either $\sigma L' \cap \text{nf}(\sigma O)$ is nonempty or $\sigma D' \cap \text{nf}(\sigma A')$ is nonempty (in order to avoid trivial state transitions).

Then we say that the state description (σL,σD) is *reachable* from the state description ($\sigma L'$,$\sigma D'$) *via the rule* R. We say that P' is reachable from P via the rules R_1, ..., R_k if there is a sequence of state descriptions P_0, ..., P_k, such that P' is reachable from P_k via R_k, and so on.

Note that, in our description of how we use a rule to move from one state description to another, we required that the input be interpreted as irreducible words, but not necessarily the output. This is because, as we noted before, we think of the input of the rule as giving a state description that must hold for the rule to fire, and of the output of the rule as giving the method for producing a new state description, but not the state description itself.

As an example of moving from one state description to another via a rule, suppose the penetrator knows the word "message" and *KEYSTATE*(a) = key1. If we let σ be the substitution Y := message, Z := a, and X := key1, then we can apply the rule

$\boxed{\text{If Y} \subseteq \mathbf{W} \text{ and } \mathit{KEYSTATE}(\text{Z}) = \text{X then } \mathbf{W} := \mathbf{W} \cup \text{e(X,Y)}}$

so that the new state becomes one in which the penetrator knows the words {message, e(key1,message)} and *KEYSTATE*(a) = key1. Thus we moved from the state description ({message},{*KEYSTATE*(a) = key1}) to the state description ({message,e(key1,message)},{*KEYSTATE*(a) = key1}) via application of the rule.

Thus we see how we can progress from state to state in following the progress of a protocol. But, in proving that a protocol is secure, we want to show, not how we can proceed from state to state, but that certain states cannot be reached from any initial state. In order to do this, we need to be able to determine, given a state description, what states can immediately precede a state satisfying that description. We do this by unifying a state description with the reduced form of the conclusion of a rule. But the techniques we apply are somewhat different than the techniques we apply in order to find what states can immediately follow a state. Suppose as before that we are given a state description (L,D). We now look for a substitution σ such that $\sigma A'$ *reduces to* a statement that does not contradict D and either some subset of σO *reduces to* a subset of σL or some subset of $\sigma A'$ *reduces to* a subset of $\sigma(D)$. We then examine σO and σA and check that

a σI, σA, σL, and σD are irreducible;

b Cond(σI,σA) is not contradicted, and;

c if σD contains an assignment $\sigma(S_i = D_i)$ and σA contradicts $\sigma(S_i = D_i)$, then nf($\sigma A'$) contains an assignment $\sigma(S_i = D_i)$; otherwise, nf(σ(A')) fails to contradict $\sigma(S_i = D_i)$.

The description of the state immediately preceding is then one in which the penetrator knows $\sigma I \cup (\sigma L$ - nf(σO), and σA holds together with all $S_i = D_i$ from σD such that S_i is not assigned a value by nf($\sigma A'$).

For example, suppose that we want to find all states immediately preceding one in which the penetrator knows the word Q, where Q is a variable standing for any word. If we let σ be the substitution Q := e(X,Y) we can apply the rule

$$\boxed{\text{If Y} \subseteq \mathbf{W} \text{ and } \mathit{KEYSTATE}\text{(Z)} = \text{X then } \mathbf{W} := \mathbf{W} \cup \text{e(X,Y)}}.$$

to get the previous state to be one in which the penetrator knows Y and *KEYSTATE*(Z) = X. On the other hand, if we apply the same rule but let σ be the substitution Y := d(X,Q), then e(X,Y) := e(X,d(X,Q)) reduces to Q and the previous state is one in which the penetrator knows d(X,Q) and *KEYSTATE*(Z) = X.

As we see from above, there is often more than one σ unifying a system state description with the conclusion of a rule. Since we want to be able to describe all possible system states that can immediately precede a state satisfying a given description, we want to be able, for each rule, to find all σ matching up the state description with the conclusion of that rule. There are usually an infinite number of such σ, but it is often possible to find a finite and *complete* set of such σ, that is a set Σ such that, if tau is a substitution matching up the state with a conclusion of a rule, then there is a $\sigma \in \Sigma$ such that $tau = \mu sigma$ for some substitution μ. (However, we will not use any members of Σ that make any terms in the target state reducible.)

We can find such complete sets by using algorithms developed for finding complete sets of unifiers with respect to some equational theory [11]. The term-rewriting system we use is Noetherian and locally confluent and thus we were able to make use of one of a class of algorithms, called *narrowing algorithms*, first discussed by Slagle [29], that can be used to generate complete sets of unifiers with respect to such term-rewriting systems. Accordingly, we have written a program in Prolog [6], that uses a modified version of the NARROWER algorithm of Rety et al. [27] to generate from a protocol specification and a description of a system state P a complete description of all system states that can immediately precede a state that satisfies P.

In order to see how the program works, suppose that we we are given a specification consisting of the single rule

$$\boxed{\text{If Y} \subseteq \mathbf{W} \text{ and } \mathit{KEYSTATE}\text{(Z)} = \text{X then } \mathbf{W} := \mathbf{W} \cup \text{e(X,Y)}}$$

and we ask the program how the penetrator can find the word Q. The program finds a complete set of unifiers of Q with e(X,Y) with respect to the term rewriting system

e(A,d(A,B)) $\rightarrow$ B
d(A,e(A,B)) $\rightarrow$ B

There are two members of the set, Q := e(X,Y), and Y := d(X,Q). Applying the first unifier, it produces the conditions Y $\subseteq$ **W** and *KEYSTATE*(Z) = X, and says the the word Q = e(X,Y) was learned. Applying the second unifier and taking the appropriate rewrites, it produces the condition d(X,Q) $\subseteq$ **W** and *KEYSTATE*(Z) = X, and says that the word Q was learned.

In order to apply our program, we want to be able to use it to show that some insecure state S defined in terms of words known by the penetrator and values of state variables is not reachable from an initial state. In order to do this, we will want to be able to show that, as we work backwards from S, we enter an infinite loop of states, none of which are the initial state. But it is not usually the case that we keep entering the same set of states over and over again. Instead, what often happens is something like the following. Suppose that we want to know whether or not the penetrator can obtain a word a. We put the question to the analyzer, and we find that it is necessary and sufficient to have obtained either the word e(X,a) and X, or d(X,a) and X, where X can be any word. When we ask the analyzer how the penetrator can obtain e(X,a) or d(X,a), we find that, in the first case, knowledge of e(Y,e(X,a)) or d(Y,e(X,a)) is needed, and in the second case, that knowledge of e(Y,e(X,a)) or d(Y,e(X,a)) is needed. These results suggest that we attempt to prove the unobtainability of the irreducible members of the formal language **K** defined by

$\mathbf{K} \rightarrow$ a
$\mathbf{K} \rightarrow$ e($\mathbf{L}$,$\mathbf{K}$)
$\mathbf{K} \rightarrow$ d($\mathbf{L}$,$\mathbf{K}$)

where $\mathbf{L}$ is the language consisting of all words of $T(\Phi,F)$, by showing that knowledge of any irreducible word of $\mathbf{K}$ requires previous knowledge of some other irreducible word of $\mathbf{K}$. We do this by running the analyzer on each production $\mathbf{K}$. We have already run it on the first, and obtained the result for that production. We next run it on e(A,B) and d(A,B), where B is assumed to be an irreducible member of $\mathbf{K}$, and analyze the results. For example, suppose that we run the analyzer on e(A,B), with the assumption that B is an irreducible member of $\mathbf{K}$, and it tells us that prior knowledge of either A and B, e(Z,e(A,B)) for some Z, or d(Z,e(A,B)) for some Z, is required. Since B, e(Z,e(A,B)), and d(Z,e(A,B)) all belong to $\mathbf{K}$, and since by the assumption under which the analyzer operates, they are all irreducible, this means that knowledge of an irreducible member of the second production of $\mathbf{K}$ requires previous knowledge of an irreducible member of $\mathbf{K}$.

Once we have proved that any state that includes knowledge of a particular word or a member of a family of words is unreachable, we can use that result to prove that other states are unreachable by showing that, in order to reach these states, the penetrator needed to know one of the unreachable words. This is the way in which our analysis usually proceeded.

5 Modeling the Selective Broadcast Protocol

In this section we describe how the selective broadcast protocol was formally modeled, and give the specification of the protocol.

We will assume that the system consists of an unspecified number of users, where each user is represented by a processor T capable of generating an infinite number of messages mess(T,N) to be encrypted, where N ranges from 1 to infinity. (Since we do not care about the content of the messages, no loss of generality results from assuming that the messages are generated by the processor rather than the user.) We will assume that messages are generated in numerical order. We only pay attention to the first K bits of each message where K is the block length of the cryptosystem involved; thus we can assume that each message is a K bit block. We will assume that the penetrator has gained control of an unspecified number of processors. We identify each processor under control of the penetrator by q(ID,dishonest), where ID is a processor identifier, one of id_1 through id_k. We identify each processor not under control of the penetrator by q(ID,honest).

We give each processor five state variables. Each state variable takes on a term or a list of terms as a value. Each state variable is subscripted with the identifier of the processor it belongs to. The state variables are listed below:

MESSTATE(T) This holds the identifier of the next message that T can generate. Initially, the variable *MESSTATE*(T) is 1 (or s(zero), where s denotes the successor operator).

IDSTATE(T) This holds the identifier of a message to be decrypted by the user and the identifier of the originator of the message.

KEYSTATE(T) holds the value of the decrypted key.

READYSTATE(T) This variable is set equal to “true” if *IDSTATE*(T) and *KEYSTATE*(T) are tested and accepted according to the rules of the protocol.

RUNSTATE(T) holds the value of the decrypted message.

The function symbols used in the modeling of the selective broadcast protocol are listed below:

name	**arity**
=	=
zero	zero
yes	zero
s	one
p	one
e	two
d	two
q	two
honest	zero
dishonest	zero
name	two
identical	two
id1, ..., idk	zero
mess	two
m	zero

We make use of the following three rewrite rules:

e(X,d(X,Y)) $\rightarrow$ Y
d(X,e(X,Y)) $\rightarrow$ Y
id_check(X,X) $\rightarrow$ yes

Lemma 4.1: The term rewriting system

e(X,d(X,Y)) $\rightarrow$ Y
d(X,e(X,Y)) $\rightarrow$ Y
id_check(X,X) $\rightarrow$ yes

is Noetherian and locally confluent.

Proof: It is clear that the system is Noetherian, since every reduction transforms a string into a shorter string. Thus every sequence of reductions is terminating. In order to prove local confluence, we make use of the Knuth-Bendix theorem [14], which uses the set of *critical pairs*, which are defined as follows:

1. Rename variables so that no two reduction rules have any variables in common.
2. For each pair of reduction rules $\lambda \rightarrow \rho$ and $\lambda' \rightarrow \rho'$ such that there is a non-variable subterm μ of λ unifying with λ', let σ be the most general unifier of μ and λ'.
3. We say that the pair $\langle \sigma\lambda[\mu \leftarrow \rho'], \sigma\rho \rangle$ is a critical pair (where $\lambda[\mu \leftarrow \rho']$ denotes the replacement of μ in λ with ρ').

The Knuth-Bendix theorem says that a rewrite system is locally confluent if, for each critical pair $\langle T_1, T_2 \rangle$, T_1 and T_2 reduce to the same unique irreducible term.

Our system contains two critical pairs, constructed from the rewrite rules

e(X,d(X,Y)) $\rightarrow$ Y
d(A,e(A,B)) $\rightarrow$ B

To obtain the first, we unify d(A,e(A,B)) with d(X,Y), to get the substitution X := A and Y := e(A,B). The critical pair thus obtained is $\langle$e(A,B),e(A,B)$\rangle$. The other pair $\langle$d(X,Y),d(X,Y)$\rangle$, is obtained by unifying e(X,d(X,Y)) with e(A,B). Since both pairs consist of two identical irreducible elements, they trivially satisfy the conditions of the Knuth-Bendix theorem. $\Box$

Now that we have shown that our term-rewriting system is Noetherian and locally confluent, we are ready to give our specification of the protocol. We first model the behavior of the honest processors and their owners, who we assume behave according to the rules of the protocol. Each honest processor can generate and encrypt messages. Each message is identified by a name denoted by name(q(X,Y),N) where q(X,Y) is the identifier of the processor originating the message, and the message is the N'th message generated by that processor. The message itself (as opposed to its name) is denoted by mess(q(X,Y),N). The distinction between message and name is that the name can be made public, while the message remains secret. Thus the rule for generation and encryption of messages is given as:

> If *MESSTATE*(q(ID,honest)) = N
>
> then $\mathbf{W} := \mathbf{W} \cup \{$name(q(ID,honest),N),
>
> e((d,q(ID,honest),d(k,name(q(ID,honest),J))),mess(q(ID,honest),N))$\}$
>
> and *MESSTATE*(q(ID),honest)) := s(J)

where s(N) denotes the successor of N.

Each honest processor can also encrypt a key for a message and send it to another processor P. We will assume that an encrypted key for message name(q(ID,honest),N) is sent only after a request for that message has been received from P. A processor (or its owner) may decide not to honor a request; however, we do not specify the decision process. Thus this rule may be thought of as a nondeterministic one. We will also assume that is is possible that a processor may forward a key without determining whether or not the identifier given is an identifier for an existing processor and not some other K-bit word.

> If $\{X,Y\} \subseteq \mathbf{W}$
>
> then $\mathbf{W}$:=
>
> $\mathbf{W} \cup \{$e(k,q(ID,honest)), e(k,e(d(k,X),d(q(ID,honest),e(d(k,X),d(q(ID,honest),d(k,Y))))))$\}$

Finally, a processor can attempt to decrypt a message. In our specification of this rule, we divided it up into four parts. Since the encrypted message is generated separately from the encrypted key, we found it simpler to specify the parts in which the key and message are loaded in two separate rules. We then included a rule in which the checking of key and message is performed, and a rule which describes the processor's behavior after the check is performed. Thus the four steps are as follows.

1. A processor receives a message name and the identifier of the sender of the message.
2. The processor receives and decrypts a key.
3. The processor checks the key against the identifiers.
4. If the check in Step 3 succeeds, the processor receives the message and decrypts it.

The separation between Steps 3 and 4 was necessary, since Step 4 involves a comparison with the reduced form of a word produced in Step 3. As our system stands now, such comparisons cannot be done within a single rule, but must be divided across two or more rules: one which generates the word, and one which does the comparison. The separation between Steps 1, 2, and 3, however, was largely a matter of convenience and was motivated by the desire to avoid having to specify a step in which the originator of an encrypted message concatenates encrypted message, encrypted key, and message identifer into a single message. Note that all we have done is break a process that was originally described in a single rule into four components. Thus there is no loss of realism.

We begin by specifying the first rule. First, a processor accepts a message name and the identifier of the message originator. In doing so, it checks that *IDSTATE* does not hold any old name-identifier pairs, that is, that *IDSTATE* = zero, and that *KEYSTATE*, *RUNSTATE*, and *READYSTATE* are zero.

If $\{X,Y\} \subseteq \mathbf{W}$
and $IDSTATE$(q(ID,honest)) = zero
and $KEYSTATE$(q(ID,honest) = zero
and $READYSTATE$(q(ID,honest)) = zero
and $RUNSTATE$(q(ID,honest)) = zero
then $IDSTATE$(q(ID,honest)) := [X,d(k,Y)].

Next, the processor accepts and decrypts a key. Again, it checks that *KEYSTATE*, *READYSTATE*, and *RUNSTATE* do not hold any old key values.

If $\{X\} \subseteq \{\mathbf{W}\}$
and $IDSTATE$(q(ID,honest)) = [Y, Z]
and $KEYSTATE$(q(ID,honest)) = zero
and $RUNSTATE$(q(ID,honest)) = zero
and $READYSTATE$(q(ID,honest)) = zero
then $KEYSTATE$(q(ID,honest)) :=
[d(q(ID,honest),e(Z,d(q(ID,honest),d(k,X))))]

Once key and identifier are loaded, the processor checks to see if they agree.

If $IDSTATE$(q(ID,honest)) = [X,Y]
and $KEYSTATE$(q(ID,honest)) = [Z]
and $READYSTATE$(q(ID,honest)) = zero
and $RUNSTATE$(q(ID,honest)) = zero
then $READYSTATE$:= [id_check(X, e(k,e(Y,Z)))]

The processor now decrypts the encrypted message. In doing so, it first checks that no message is presently stored in *RUNSTATE*.

If X $\subseteq \mathbf{W}$
and $KEYSTATE$(q(ID,honest)) = [Y]
and $READYSTATE$(q(ID,honest)) = [yes]
and $RUNSTATE$(q(ID,honest)) = zero
then $RUNSTATE$(q(ID,honest)) := [d(Y,X)]

The processor can also zero out all variables at any time. In an actual application, this would either be at the direction of the user, or possibly in response to an error, or both. Note that, since we do not specify the conditions under which the zeroing out may occur, the "if" portion of this rule is empty.

$IDSTATE$(q(ID,honest)) = zero
and $KEYSTATE$(q(ID,honest)) = zero
and $READYSTATE$ (q(ID,honest)) = zero
and $RUNSTATE$(q(ID,honest)) = zero

We now model the interactions of a penetrator with the processors under his control. First of all, the penetrator can use a terminal to encrypt a message. This is similar to the way in which an honest terminal encrypts a message, except the values encrypted can be any arbitrary values input by the penetrator:

If {X,Y,Z} $\subseteq$ **W**
then **W** := **W** $\cup$ {e(k,q(ID,dishonest)),
e(k,e(d(k,Y),d(q(ID,dishonest),e(d(k,Y),d(q(ID,dishonest),d(k,X)))))),
e(d(q(ID,dishonest),e(d(k,Y),e(d(q(ID,dishonest),d(k,X)))),Z))}

The rule for encrypting a key is identical to the rule for encrypting a message, except that the last output is not given. Since the penetrator learns no more from that rule than from the rule for encrypting a message, we will omit it.

Secondly, the penetrator can use a terminal in order to attempt to decrypt a message. These rules are identical to the rules for the honest processors, except that, in the rule describing *READYSTATE*, the penetrator can learn the terminal output. The *READYSTATE* rule is given below:

If *IDSTATE*(q(ID,dishonest)) = [X,Y]
and *KEYSTATE*(q(ID,dishonest)) = [Z]
and *READYSTATE*(q(ID,dishonest)) = zero
then *READYSTATE*(q(ID,dishonest)) := [id_check(X, e(k,e(Y,Z)))]
and **W** := **W** $\cup$ {e(k,q(ID,dishonest)), e(k,e(Y,Z)))}.

Finally, we assume that the penetrator can encrypt and decrypt data on his own:

If {X,Y} $\subseteq$ **W** then **W** := **W** $\cup$ {e(X,Y)}
If {X,Y} $\subseteq$ **W** then **W** := **W** $\cup$ {d(X,Y)}

There are at least three goals that a penetrator may try to attain. One of these, of course, is to decrypt a message from an honest processor on one of the processors under his control that is not authorized to decrypt that message. Note that this includes the case in which another processor under his control may be authorized to decrypt the message. The second is to convince a processor to decrypt a message supplied by the penetrator under the illusion that it is decrypting a message supplied by an honest processor. Finally, if the protocol is used in a situation in which users are not allowed to see messages in the clear even if they are allowed to decrypt them inside their processor, the penetrator may try to obtain a copy of an honest processors's message in the clear.

The penetrator will have attained his first goal if he is able make some state variable *RUNSTATE*(q(ID1,dishonest)) take the value mess(q(ID2,honest),K) without processor q(ID1,dishonest) having requested the right to that message. He will have attained his second goal if he is able to make one of the state variables *RUNSTATE*(q(ID2,honest)) = mess(q(ID1,dishonest),N) and *IDSTATE* = [Q,q(ID3,honest)]. He will have attained his third goal if some mess(q(ID1,honest),N) $\in$ **W**.

We will assume that initially the penetrator knows all processor identifiers e(k,q(A,B)), as well as a set of messages mess(q(ID,dishonest),B) belonging to the dishonest processors, along with their names, and the value zero.

6 Analysis of the Selective Broadcast Protocol

In our analysis of the selective broadcast protocol, we asked three questions:

1. Can a penetrator obtain a cleartext copy of a program belonging to an honest user of the system?

2. Can a module under the control of the penetrator gain access to a program belonging to an honest user which has not been legitimately been granted by that user?

3. Can a penetrator pass off his own program as a program originated by an honest user?

The answers to the first two questions turned out to be negative. Details of that analysis are provided in [18].

As we mentioned earlier, the answer to the third question was positive. But, in order to be able to present our analysis, we need to make use of several lemmas we proved about the unobtainability of certain kinds of words. We omit the proofs since they are lengthy and similar to other proofs that are provided in this paper. Details of the proofs are provided in [18].

Lemma 5.1: It is impossible for a penetrator to obtain an irreducible word of the form d(X,Y) unless X and Y are already known.

Thus, if we ask the analyzer how a penetrator could find some word Y, we can ask it to ignore all output that requires previous knowledge of d(X,Y), since this would require that Y already be known.

Lemma 5.2: Let **A** be the formal language defined by

$\mathbf{A} \rightarrow \mathrm{k}$
$\mathbf{A} \rightarrow \mathrm{e}(\mathbf{L},\mathbf{A})$
$\mathbf{A} \rightarrow \mathrm{d}(\mathbf{L},\mathbf{A})$

where **L** is the set of all possible words from $T(\Phi,\mathrm{F})$. Then it is impossible for the penetrator to obtain any irreducible member of **A**.

Corollary 5.3: The word d(q(ID2,honest),d(k,name(q(ID2,honest),Q))) is unobtainable.

Proof: By repeated application of Lemma 5.1, d(q(ID2,honest),d(k,name(q(ID2,honest),Q))) is unobtainable unless k is previously known. But Lemma 5.2 tells us that the word k is unobtainable. □

The next lemma requires a few preliminary definitions. To begin with, we need the standard definition of occurrence, as presented in Huet and Oppen [11]. An occurrence is a means for locating subterms in a term that makes use of a notation similar to the Dewey Decimal system.

Definition 5.4: If T is a term, we define the set of occurrences in T, O(T) and their values as follows:

1. $\epsilon \in O(\mathrm{T})$ and $\mathrm{T}/\epsilon = \mathrm{T}$. In other words, ϵ is the occurence of T in itself.

2. If $\mathrm{T} = \mathrm{f}(T_1,...,T_n)$, and $o \in O(T_i)$, then $i.o \in O(T_i)$ and $\mathrm{T}/i.o = T_i/o$.

If $o \neq \epsilon$, define *prefix(o)* to be the string o minus the last number, that is, the occurrence of the parent term of the term occurring at o.

Thus, for example, in the word T = e(a,d(b,c)), T/1 = a, T/2 = d(b,c), and T/2.1 = b.

We now define odd words. [1]

Definition 5.5: We begin by defining even words. These are all irreducible words W from $T(\Phi,\mathrm{F})$ that satisfy the following criteria:

a) There exist expressions A and N such that W is a member of the formal language **B**(A,N) defined by

[1] The definition of odd word replaces the definition of odd word in [18], which was incorrect. (The string of 2's in this definition replaces the string of 1's in the earlier definition.) However, the proof of the following lemma is identical to the proof in [18], and so we omit it here.

B(A,N) $\rightarrow$ d(k,name(q(A,honest),N))
B(A,N) $\rightarrow$ e(**L**,**B**(A,N))
B(A,N) $\rightarrow$ d(**L**,**B**(A,N))

b) There is at least one other occurrence of k at 2*.1, where 2* represents a possibly empty sequence of 2's such that e(k,Y) occurs at the prefix of that occurrence in W for some expression Y.

c) Let O be such an occurrence, and suppose there is no other such occurrence in W. Let O' be the prefix of O. Then, for all expressions E, the number of occurrences S in W of the form O'.2*.1 such that E occurs at S is even.

A word is *odd* if it satisfies the first two criteria and no instance of it satisfies the third.

Thus

e(k,e(q(A,honest),d(q(B,dishonest),d(k,name(q(ID2,honest),N)))))

is odd, since q(A,honest) occurs at 2.1 and q(B,dishonest) occurs at 2.2.1, and so is

e(q(B,dishonest),e(k,e(q(B,dishonest),d(k,name(q(ID2,honest),N))))),

since, while q(B,dishonest) occurs twice, there is only one occurrence, 2.2.1, that is of the form O'.2*.1, where O' = 2 is the prefix of O = 2.1, the other occurrence of k. However,

e(k,e(X,d(Y,e(X,d(Y,d(k,name(q(ID2,honest),N)))))))

is even, since X occurs twice at 2.1 and 2.2.2.1, and Y occurs twice at 2.2.1 and 2.2.2.2.1, where O, the other occurrence of k, is 1, and O' is ϵ. The word

e(k,e(X,e(Y,d(k,name(q(ID2,honest),N)))))

is not even, but is not odd, either, since the instance given by the substitution X = Y is even.

Lemma 5.6: No odd word is obtainable by the penetrator.

Proof: See [18].

We now wish to attempt to show that a penetrator cannot pass off his own message as somebody else's. Recall that a message mess(q(ID,dishonest),N) has been successfully substituted for a message mess(q(ID2,honest),Q) if there is a processor q(B,honest) with

IDSTATE(q(B,honest)) = [name(q(ID2,honest),Q),q(ID2,honest)]
RUNSTATE(q(B,honest)) = mess(q(ID,dishonest),N)

Accordingly we run the analyzer on these two state values and find that the following set of conditions must hold immediately prior to attaining these values:

knowledge of e(X,mess(q(ID,dishonest),N))
IDSTATE(q(B,honest)) = [name(q(ID2,honest),Q),q(ID2,honest)]
KEYSTATE(q(B,honest)) = X
READYSTATE(q(B,honest)) = yes
RUNSTATE(q(B,honest)) = zero

Running the analyzer on the four state values given above, we find that the following must have held previously:

IDSTATE(q(B,honest)) = [name(q(ID2,honest),Q),q(ID2,honest)]
KEYSTATE(q(B,honest)) = X = d(Y,d(k,name(q(ID2,honest),Q)))
READYSTATE(q(B,honest)) = zero
RUNSTATE(q(B,honest)) = zero

Thus the penetrator can substitute mess(q(ID,dishonest),N) for mess(q(ID2,honest),Q) if and only if he can obtain e(d(q(ID2,honest),d(k,name(q(ID2,honest),Q)))),mess(q(ID,dishonest),N)) and cause the state variables of processor q(B,honest) to take on the values given above.

Running the analyzer on the state values listed above, we find that one of the following is required to hold previously:

a) *IDSTATE*(q(B,honest)) = [name(q(ID2,honest),Q),q(B,honest)]
KEYSTATE(q(B,honest)) = zero
READYSTATE(q(B,honest)) = zero
RUNSTATE(q(B,honest)) = zero
with knowledge of name(q(ID2,honest),Q)

b) *IDSTATE*(q(B,honest)) = [name(q(B,honest),Q),q(ID2,honest)]
KEYSTATE(q(B,honest)) = zero
READYSTATE(q(B,honest)) = zero
RUNSTATE(q(B,honest)) = zero
with knowledge of
e(k,e(q(B,honest),
d(q(ID2,honest),e(q(B,honest),d(q(ID2,honest),d(k,name(q(ID2,honest),Q)))))))

The word name(q(ID2,honest),Q) is obtainable when processor q(ID2,honest) transmists the word name(q(ID2,honest),Q), while the word

e(k,e(q(B,honest),d(q(ID2,honest),e(q(B,honest),d(q(ID2,honest),d(k,name(q(ID2,honest),Q)))))))

is obtainable by having processor q(B,honest) purchase the rights to mess(q(ID2,honest),Q).

Running the analyzer on the state values given in a) and b), we are told that they may be obtained if previously all state values are zero and name(q(ID2,honest),Q) is previously known, and, in case a), if e(k,q(B,honest)) is previously known, and, in case b), if e(k,q(ID2,honest)) is known. Since all e(k,q(S,T)) are assumed to be known, and since all state values can be made zero at any time by applying the rule that sets all the values to zero, and since name(q(ID2,honest),Q) is obtainable, this set of state values is obtainable. Thus, working our way back up, we have shown that the set of state values

IDSTATE(q(B,honest)) = [name(q(ID2,Q),q(ID2,honest)]
RUNSTATE(q(B,honest)) = mess(q(ID,dishonest),N)

is obtainable if and only if the word

Z = e(d(q(ID2,honest),d(k,name(q(ID2,honest),Q))),mess(q(ID,dishonest),N))

is obtainable.

Thus our next task is to show that Z is unobtainable, and we set out to do this. However, after running the analyzer on Z we find that this it *is* obtainable by the penetrator. The scenario is as follows. Processor q(ID,dishonest) obtains an encrypted key to mess(q(ID2,honest),Q) from processor q(ID2,honest). Processor q(ID,dishonest) applies the rule for encrypting messages:

If $\{A,B,M\} \subseteq \mathbf{W}$
then $\mathbf{W} := \mathbf{W} \cup \{$e(k,q(ID,dishonest)),
e(k,e(d(k,B),d(q(ID,dishonest),e(d(k,B),d(q(ID,dishonest),d(k,A)))))),
e(d(q(ID,dishonest),e(d(k,B),e(d(q(ID,dishonest),d(k,A)))),M))$\}$

with input

A = e(k,e(q(ID,dishonest),
    d(q(ID2,honest),e(q(ID,dishonest),d(q(ID2,honest),d(k,name(q(ID2,honest),Q)))))))
B = e(k,q(ID2,honest))
M = mess(q(ID,dishonest),N)

where A is the encrypted key to mess(q(ID,honest),Q). The word output is

Z = e(d(q(ID2,honest),d(k,name(q(ID2,honest),Q))),mess(q(ID,dishonest),N)).

This gives us the penetration scenario described in Section 2.

Fortunately, the flaw uncovered is easy to correct. If the rule for encrypting messages had been applied correctly, the second word input would have been e(k,q(ID,dishonest)). Moreover, both k and q(ID,dishonest) are stored in processor q(ID,dishonest). Thus, we can alter the message encryption rule such that the processor either checks that the second word input is the correct one, or supplies the word itself without requiring any input. Thus we can replace the penetrator's program encryption rule with

If $\{$P M$\} \subseteq \mathbf{W}$
then $\mathbf{W} := \mathbf{W} \cup \{$e(k,q(ID,dishonest)),P,e(d(q(ID,dishonest),d(k,P)),M)$\}$

However, recall that the original message encryption rule subsumed the key encryption rule as well. This is no longer the case, and so we must supply the key encryption rule separately:

If $\{$A,B$\} \subseteq \mathbf{W}$
then $\mathbf{W} := \mathbf{W} \cup \{$e(k,q(ID,dishonest)),
e(k,e(d(k,B),d(q(ID,dishonest),e(d(k,B),d(q(ID,dishonest),d(k,A))))))$\}$.

We now wish to prove that the new protocol satisfies the desired security property. However, we do not wish to reverify the properties the old protocol already satisfies if we can avoid it. Thus we introduce the notion of *restrictions* of rules and protocols.

Definition 5.7: A rule R_2 =

If $I_2 \subseteq \mathbf{W}$ and A_2 and Cond(I_2,A_2) then $\mathbf{W} := \mathbf{W} \cup O_2$ and A'_2

is a restriction of a rule R_1 =

If $I_1 \subseteq \mathbf{W}$ and A_1 and Cond(I_1,A_1) then $\mathbf{W} := \mathbf{W} \cup O_1$ and A'_1

if there is a substitution σ acting as the identity on the variables of R_2 such that

1. $\sigma I_1 \subseteq \sigma I_2$;
2. σA_2 implies σA_1;
3. σCond(I_2,A_2) implies σCond(I_1,A_1);
4. $\sigma O_2 \subseteq \sigma O_1$, and;
5. $\sigma A'_1 = \sigma A'_2$.

Definition 5.8: A protocol specification $\mathbf{S}'$ is a restriction of a protocol specification $\mathbf{S}$ if $\mathbf{S}'$ is obtained by replacing every rule in $\mathbf{S}$ either by itself or by a number of restrictions of that rule.

Lemma 5.9: Let **S** be a protocol specification, and let **S**' be a restriction of **S**. Let P be a description of a state in terms of words known by a penetrator and values of state variables. If all states described by P are unreachable in **S**, then all such states are unreachable in **S**'. Moreover, if the states described by P are unreachable in **S** except via some state described by a description P', then the same is true for **S**'.

Proof: Let P be such a description, and suppose that there is a state Q described by P that is reachable in **S**' but not in **S**. Thus there is a sequence

$Q_0 \to R_0 \to Q_1 \to .... \; R_{k-1} \to Q_k \to R_k \to$ Q

where Q_0 is an initial state, and $Q_i \to R_i \to Q_{i+1}$ means that Q_{i+1} is reached from Q_i by means of R_i. Since each R_i is either equal to a rule R_i' in **S** or a restriction of such a rule, there exists a substitution σ on the variables of R_i' acting as the identity on the variables of R_i such that R_i' differs from σR_i only in that possibly more words are required for input, fewer words are output, and stronger conditions are required on the input words and states. Thus, if Q_{i+1} is reachable from Q_i via R_i', it is reachable via σR_i, and hence, by the definition of reachability, via R_i. Thus Q is reachable in $\mathbf{S}'$.

A similar argument works for the second assertion of this lemma. □

The two new rules that we defined satisfy all the requirements of a restriction of the original rule except that the input words of the original rule do not form a subset of the input words of the new rules. We can get around this noting that this property is satisfied if we require that e(k,q(ID,dishonest)) is required to be an input word in the program encryption rule. Since this word is assumed to be known initially by the penetrator, its inclusion or omission has no effect on the reachability or nonreachability of a state. Thus we do not need to reprove any of the lemmas that we proved earlier. Nor do we need to reprove the fact that a penetrator can pass off his own message as somebody else's only if he is able to produce the word e(d(q(ID2,honest),d(k,name(q(ID2,honest),Q))),mess(q(ID,dishonest),N))

We now want to use our revised specification to prove the theorem:

Theorem 5.10: The word

e(d(q(ID2,honest),d(k,name(q(ID2,honest),Q))),mess(q(ID,dishonest),N))

is unobtainable.

Proof: We now run the analyzer using the revised specification, on the word

e(d(q(ID2,honest),d(k,name(q(ID2,honest),Q))),mess(q(ID,dishonest),N)),

asking it to ignore cases that require prior knowledge of

d(Z,e(d(q(ID2,honest),d(k,name(q(ID2,honest),Q))),mess(q(ID,dishonest),N)))

or

d(q(ID2,honest),d(k,name(q(ID2,honest),Q))),

and, after discarding cases which require prior knowledge of an odd word, we find that one of the following three conditions must be satisfied:

a) prior knowledge of
e(X,e(d(q(ID2,honest),d(k,name(q(ID2,honest),Q))),mess(q(ID,dishonest),N)))
and X;

b) prior knowledge of

e(k,e(B,d(C,e(B,d(C,d(k,e(d(q(ID2,honest),d(k,name(q(ID2,honest),Q))),
mess(q(ID,dishonest),N)))))))),
or;

c) *KEYSTATE*(q(E,dishonest)) =
d(X,d(k,e(d(q(ID2,honest),d(k,name(q(ID2,honest),Q))),mess(q(ID,dishonest),N))))
and *IDSTATE*(q(E,dishonest)) = [Y,X].

In the case of c, we run the analyzer on *KEYSTATE*(q(E,F)) =

e(X,d(k,e(d(q(ID2,honest),d(k,name(q(ID2,honest),Q))),mess(q(ID,dishonest),N))))

and *IDSTATE*(q(E,F)) = [Y,X], and find that prior knowledge of one of

c1) e(k,e(B,d(C,e(B,d(C,d(k,
e(d(q(A,honest),d(k,name(q(ID2,honest),Q))),mess(q(ID,dishonest),N)))))))), or;

c2) e(d(q(ID2,honest),d(k,name(q(ID2,honest),Q))),mess(q(ID,dishonest),N))

is required. This leads us to attempt to verify the unobtainability of all irreducible words from the language **C** defined by

C $\rightarrow$ e(d(q(ID2,honest,d(k,name(q(ID2,honest),Q))),mess(q(ID,dishonest),N))
C $\rightarrow$ e(**L**,**C**)
C $\rightarrow$ d(**L**,**C**)

Lemma 5.11: No irreducible word of **C** is obtainable.

Proof: We prove that **C** is unobtainable by showing that knowledge of a word from **C** implies previous knowledge of either a word already shown to be unobtainable or another word from **C**. We do this by using the technique outlined in Section 2, and running the analyzer on each of the productions of **C**. However, by Lemma 5.1, knowledge of d(X,Y) implies previous knowledge of Y, so we already know that knowledge of a word from the third production implies previous knowledge of a word from **C**. Thus we only need to run the analyzer on the first two productions. We first run it on

X = e(d(q(ID2,honest,d(k,name(q(ID2,honest),Q))),mess(q(ID,dishonest),N))

asking it to ignore all solutions that require previous knowledge of X, e(Z,X), or d(Z,X) for some Z, since all these words are known to be members of **C**. The analyzer produces a number of solutions. All of them require previous knowledge of a word of **C**, except for two, one of which requires previous knowledge of

e(k,e(q(ID3,dishonest),d(q(ID2,honest),d(k,name(q(ID2,honest),N))))).

and one of which requires previous knowledge of

mess(q(ID,dishonest),N)
and
d(q(ID2,honest),d(k,name(q(ID2,honest),N))).

The last of these is unobtainable by Corollary 5.3. Next, we run the analyzer on e(X,Y), where Y is assumed to be a member of **C**. We ask the analyzer to ignore all solutions that require previous knowledge of e(X,Y), Y, e(Z,e(X,Y)), or d(Z,e(X,Y)). All of the necessary conditions turned up by the analyzer fall into one of the following cases:

a) X = k, Y = e(A,d(B,e(A,d(B,d(k,C))))), and prior knowledge of C is required;
b) X = k, Y = e(k,e(A,d(B,e(A,d(B,C))))), and prior knowledge of e(k,C) is required;
c) X = k, Y = e(A,d(B,e(A,C))), and prior knowledge of e(k,e(B,C)) and e(k,B) is required;
d) X = k, Y = e(A,d(B,C)), and prior knowledge of e(k,e(B,d(A,C))) is required;
e) X = k, Y = e(A,C), and prior knowledge of e(k,B) and e(k,e(B,d(A,e(B,C)))) is required;
f) X = k, Y = C, and prior knowledge of e(k,e(B,d(A,e(B,d(A,C))))) is required;
g) X = D, Y = C, and prior knowledge of e(k,e(B,d(A,e(B,d(A,d(k,e(D,C))))))) is required;
h) X = k, Y = d(q(R,dishonest),e(A,C))), and prior knowledge of e(k,e(q(R,dishonest),C) is required.

We begin by giving the proof for case a) in detail. Since we assumed Y ∈ **C**, and Y = e(A,d(B,e(A,d(B,d(k,C))))), this means that either d(B,e(A,d(B,d(k,C))))) ∈ **C** or A = d(q(ID2,honest,d(k,name(q(ID2,honest),Q))) and d(B,e(A,d(B,d(k,C)))) reduces to the word mess(q(ID,dishonest),N). But the latter is impossible, since Y, and hence d(B,e(A,d(B,d(k,C)))), is assumed to be irreducible. Thus, d(B,e(A,d(B,d(k,C)))) ∈ **C**. From this fact, by the definition of **C** and the irreducibility of the word d(B,e(A,d(B,d(k,C)))), we have e(A,d(B,d(k,C))) ∈ **C**. Using this fact and reasoning similar to the case for the word d(B,e(A,d(B,d(k,C)))), we get that d(B,d(k,C)) ∈ **C**. From the definition of **C**, we get that d(k,C) ∈ **C**, and using the definition of **C** again, we get that C ∈ **C**. Since prior knowledge of C is required in case a), this means that prior knowledge of a word from **C** is required.

Similar reasoning can be used to prove the result for all cases except c), e), and h); we show that the word must belong to the second or third production of **C**, and work our way down.

In cases c), e) and h) this reasoning fails if A = d(q(ID2,honest),d(k,name(q(ID2,honest),N))) and C = mess(q(ID,dishonest),N). In cases c) and e), if these values for A and C hold, prior knowledge of

e(k,A) = e(k,d(q(ID2,honest),d(k,name(q(ID2,honest),N))))

is required. In case h), if these values for A and C hold, prior knowledge of

e(k,q(R,dishonest),C)) = e(k,q(R,dishonest),mess(q(ID,dishonest),N))

is required. Thus, all we have to show is that

e(k,d(q(ID2,honest),d(k,name(q(ID2,honest),N)))),
e(k,e(q(ID3,dishonest),d(q(ID2,honest),d(k,name(q(ID2,honest),N))))),
and
e(k,q(R,dishonest),mess(q(ID,dishonest),N))

are unobtainable. But these are all odd words, which are unobtainable by Lemma 5.6. Thus we have shown that knowledge of a member of the language **C** requires prior knowledge of an unobtainable word or of another member of **C**, and so we are done. □

7 Discussion

The fact that our techniques could be used to find security flaws in a previously published protocol shows that it is likely that they can be refined and developed into useful tools for protocol analysis. Our experience in specifying and analyzing this protocol also points out ways in which they can be improved.

To begin with, in our specification we modeled a protocol in which all systems operated correctly. Thus we were able to show that no security violations occurred in a system in which no errors occurred. However, the goal of most protocols is, not only to ensure security if everything operates correctly, but to minimize the effects of system errors and security violations. If it fails to do so, it is considered to be inadequate. For example, the flaw in the Needham-Schroeder protocol [23] discovered by Denning and Sacco [7] required the existence of a previously compromised session key to be exploited.

Similar issues arose in our analysis of the Simmons protocol in [18]. For example, we discovered that it was possible for a penetrator to gain illegal access to messages if the cryptographic modules failed in a certain way. Although in [18] we listed this as a flaw in the protocol, it can not really be considered one since the Simmons protocol explicitly depends upon the correct operations of the cryptographic modules. However, it is still of interest because the type of failure required was not one that would appear at first glance to be likely to cause a security breach. Hence, knowledge of the effects of this failure might be useful to someone who is trying to find out what assumptions about the cryptographic modules we can afford to weaken.

Thus, there are two types of failures that it would be useful to be able to handle. One of these is the class of failures that we consider likely to occur, such as the compromise of a used session key. We can include such failures by specifying them as part of the protocol; this, for example, is what we did in [19] in an analysis of a variant of the Needham-Schroeder protocol. The other is the class of failures that may or may not be likely, but can be shown to cause security failures in the protocol. It is, of course, impossible to list all such failures, but it would be useful to have a system that could assist in their discovery.

Another related question that arose in our analysis was the question of what penetrator actions we should consider relevant to the security of the protocol. Clearly, the ability of the penetrator to insert information into his own processors, to communicate with the other protocol participants, and to encrypt and decrypt on his own, were relevant to the security of the protocol and were included in the specification, but we did not include the penetrator's ability to apply the successor and predecessor operators. However, we had no formal basis for making these decisions. Likewise, if we had also attempted to specify the incorrect operation of the protocol, we would have had no good means of choosing which errors to consider as important.

One means of deciding what penetrator actions to include would be to examine previous, similar protocols and see what actions penetrators used to exploit flaws in these. This was the reasoning behind our decision to include the actions we did; we looked at the earlier protocol described in [25] and included the actions that a penetrator would have used to exploit its security flaw. But it would be helpful if we had a more formal basis for excluding certain penetrator actions in the analysis of the protocol. In particular, we would like to be able to prove general results that give classes of protocols such that, if a protocol belongs to a certain class and is vulnerable to an attack involving a certain penetrator action, then it is vulnerable to an attack not involving that action. This is the kind of result proved by Even, Goldreich, and Shamir in [9]. They define a class of public-key protocols and show that, if a protocol belonging to this class is vulnerable to an attack involving certain algebraic properties of the RSA, then it is vulnerable to an attack not involving these properties. The result of Even et al. was obtained for a relatively narrowly defined class of protocols, and it is unlikely that we would be able to prove similar theorems for more broadly defined classes. However, we might be able to develop techniques for proving theorems about protocols that will allow us to show that a particular penetrator action is irrelevant to the security of a given protocol.

Another question that arises is: how do we characterize the insecure states of a protocol? The techniques that we have outlined in this paper give us no assistance in answering this question. Our techniques allow us to specify the operation of a protocol, but they do not help us to determine its requirements. Extensions to the our system that we have subsequently developed make characterization of security and insecurity a little easier; in [19] we introduce a model in which we can specify the security of a protocol not only in terms of words learned by a penetrator and values of internal state variables, but in terms of histories of events that should or should not occur. Since most cryptographic protocols are informally specified in terms of such histories, this allows a more natural way of mapping the designer's intuitive notion of what characterizes the desirable behavior of a protocol to our more formal model. However, it still does not give a formal methodology for characterizing the security of a protocol. In order to do so, we should turn to other methods. One such is the use of logics of belief or trust to analyze cryptographic protocols, such as the work of Burrows, Abadi, and Needham [4], or that of Rangan [26], both discussed in Section 2. In these systems, a protocol is mapped to a set of assertions in modal logic. The goal of the protocol is mapped to another such assertion. The set of assertions is then analyzed using formally defined inference rules to determine whether or not the asserted goal of the protocol is derivable. If they are not, the assertions that the protocol should but fails to provide are generated. Thus the security of a protocol is examined at a much higher level of abstraction then our method of analysis provides.

The weakness of this approach is that the mapping of the protocol to the assertions and the construction of the inference rules is relatively informal. Moreover, since the level of abstraction is high, many features of the protocol are necessarily not captured. Thus, although the use of these techniques in the analysis of the selective broadcast protocol would probably have uncovered the need for the assertion that a participant in the protocol can only encrypt messages with his own key, it would have been unlikely to provide any assistance in determining whether or not the protocol satisfied this assertion. Our analysis showed that the protocol did not.

Thus, it seems that what might be useful is some sort of layered approach, applying modal logic techniques to determine the requirements of a protocol, and then using techniques similar to those described in this paper to determine whether or not a protocol can meet those requirements. This is in line with current practice in system verification. There is no one verification technique that is appropriate for all layers of abstraction. Instead, one uses different techniques to perform specification and verification at different level of abstractions. The results of the verification at a given level provide assertions to be proved at the next lower level of abstraction.

One more question remains to be answered. That is: how well will these techniques scale up? Our techniques worked well on a relatively simple protocol that could be specified using a small number of protocol rules and a small number of rewrite rules. How can we make them work well on more complex protocols? This question can be answered in part by improving the mechanisms we use in the protocol analysis: by improving the efficiency of the narrowing algorithm we use, by extending the program so that it generates complete sets of equational unifiers for broader classes of equational theories, and by extending the functionality of the program. For example, it would be useful if our program included mechanisms for determining whether or not a word belonged to a formal language input by the protocol analyzer. This would have eliminated a lot of the hand analysis we had to do. But we will also need to develop techniques for showing that the composition of already verified pieces of a system satisfies its security requirements, and techniques for limiting the reverification that needs to be done when a system is modified.

8 Conclusion

In this paper we have presented a formal model for a class of cryptographic protocols, a procedure for proving security properties of protocols specified according to this model, and a program that provides automated assistance in these proofs. We specified a published protocol and attempted to prove it secure using our techniques. The fact that these techniques exposed a flaw in the protocol provides evidence that these techniques can provide significant assistance in the analysis of protocols.

9 Acknowledgements

I am grateful to Paul Syverson and Jim Gray for their careful reading of and insightful comments on earlier versions of this paper, and to Paul Syverson for illuminating discussions on the role of logics of knowledge and belief in the analysis of cryptographic protocols. I am also grateful to the referees, whose comments pointed out many points that needed clarification and greatly improved the presentation of this paper.

References

[1] R. K. Bauer, T. A. Berson, and R. J. Feiertag. A key distribution protocol using event markers. *ACM Transactions on Computer Systems*, 1:249–255, August 1983.

[2] P. Bieber. A logic of communication in a hostile environment. In *Proceedings of the Computer Security Foundations Workshop III*, pages 14–22, Franconia, NH, June 12-14 1990. IEEE Computer Society Press.

[3] D. E. Britton. Formal verification of a secure network with end-to-end encryption. In *Proceedings of the 1984 Symposium on Security and Privacy*, pages 154–166, Oakland, CA, 1984. IEEE Computer Society Press.

[4] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, February 1990.

[5] CCITT. *CCITT Draft Recommendation X.509. The Directory-Authentication Framework, Version 7*, November 1987.

[6] W. F. Clocksin and C. S. Mellish. *Programming in Prolog*. Springer-Verlag, New York, 1984.

[7] D. E. Denning and G. M. Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24(8):533–536, August 1981.

[8] D. Dolev and A. Yao. On the security of public-key protocols. *IEEE Transactions in Information Theory*, 29:198–208, 1983.

[9] S. Even, O. Goldreich, and A. Shamir. On the security of ping-pong protocols when implemented using the RSA. In *Advances in Cryptology - CRYPTO '85*, pages 58–72, Santa Barbara, CA, 1985. Springer-Verlag.

[10] L. Gong, R. Needham, and R. Yahalom. Reasoning about belief in cryptographic protocols. In *Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy*, pages 234–248, Oakland, CA, May 1990. IEEE Computer Society Press.

[11] G. Huet and D. C. Oppen. Equations and rewrite rules: A survey. In R. V. Book, editor, *Formal Language Theory: Perspectives and Open Problems*, pages 349–405. Academic Press, New York, 1980.

[12] T. Kasami, S. Yamamura, and K. Mori. A key management scheme for end-to-end encryption and a formal verification of its security. *Systems, Computers, Control*, 13:59–69, May-June 1982.

[13] R. A. Kemmerer. Using formal methods to analyze encryption protocols. *IEEE Journal on Selected Areas in Communications*, 7(4):448–457, May 1989.

[14] D. Knuth and P. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, 1970.

[15] D Longley and S. Rigby. Use of expert systems in the analysis of key management systems. In E A. Grissonnanche, editor, *Security and Protection in Information Systems*, pages 213–224. North-Holland, Amsterdam, 1989.

[16] W.-P. Lu and M. K. Sundareshan. Secure communication in internet environments: A hierarchical key management scheme for end-to-end encryption. *IEEE Transactions on Communications*, 37(10):1014–1023, October 1989.

[17] C. A. Meadows. Using narrowing in the analysis of key management protocols. In *Proceedings of the 1989 IEEE Computer Society Symposium on Security and Privacy*, pages 138–147, Oakland, CA, May 1989. IEEE Computer Society Press.

[18] C. A. Meadows. Applying formal methods to the analysis of a key management protocol. NRL Report 9265, Naval Research Laboratory, Washington, DC, September 19 1990.

[19] C. A. Meadows. A system for the specification and verification of key management protocols. In *Proceedings of the 1991 IEEE Symposium on Security and Privacy*, pages 182–195, Oakland, CA, May 1991. IEEE Computer Society Press.

[20] J. K. Millen, S. C. Clark, and S. B. Freedman. The Interrogator: Protocol security analysis. *IEEE Transactions on Software Engineering*, SE-13, No. 2, February 1987.

[21] J. H. Moore. Protocol failures in cryptosystems. *Proceedings of the IEEE*, 76(5):594–602, May 1988.

[22] L. E. Moser. A logic of knowledge and belief for reasoning about computer security. In *Proceedings of The Computer Security Foundations Workshop II*, pages 57–63. IEEE Computer Society Press, 1989.

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

[24] D. Nesset. A critique of the Burrows, Abadi, and Needham logic. *ACM Operating Systems Review*, 24(2):35–38, April 1990.

[25] G. B. Purdy, G. J. Simmons, and J. A. Studier. A software protection scheme. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 99–103, Oakland, CA, 1982. IEEE Computer Society Press.

[26] P. V. Rangan. An axiomatic basis of trust in distributed systems. In *Proceedings of the 1988 Symposium on Security and Privacy*, pages 204–211, Oakland, CA, 1988. IEEE Computer Society Press.

[27] P. Rety, C. Kirchner, H. Kirchner, and P. Lescanne. NARROWER: A new algorithm for unification and its application to logic programming. In *Rewriting Techniques and Applications, Lecture Notes in Computer Science, vol. 202*, New York, 1985. Springer-Verlag.

[28] G. J. Simmons. How to (selectively) broadcast a secret. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 108–113, Oakland, CA, 1985. IEEE Computer Society Press.

[29] J. R. Slagle. Automated theorem-proving for theories with simplifiers, commutativity, and associativity. *JACM*, 21(4):622–642, October 1974.

[30] P. Syverson. Formal semantics for logics of cryptographic protocols. In *Proceedings of the Computer Security Workshop III*, pages 32–41, Franconia, NH, June 12-14 1990. IEEE Computer Society Press.

[31] P. Syverson. A logic for the analysis of cryptographic protocols. NRL Report 9305, Naval Research Laboratory, December 31 1990.

[32] P. Syverson. The use of logic in the analysis of cryptographic protocols. In *Proceedings of the 1991 IEEE Symposium on Security and Privacy*, pages 156–170, Oakland, CA, May 1991. IEEE Computer Society Press.

[33] V. Viradharajan and S. Black. Formal specification of a secure distributed system. In *Proceedings of the 12th National Computer Security Conference*, pages 146–171, Baltimore, MD, October 1989. NCSC.

[34] K. A. Yelick. Unification in combinations of collapse-free regular theories. *Journal of Symbolic Computation*, 3:153–181, 1987.