



**Applying Formal Methods to the Analysis of a
Key Management Protocol**

C. A. MEADOWS

*Center for Secure Information Technology
Information Technology Division*

September 19, 1990

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE September 19, 1990	3. REPORT TYPE AND DATES COVERED	
4. TITLE AND SUBTITLE Applying Formal Methods to the Analysis of a Key Management Protocol		5. FUNDING NUMBERS	
6. AUTHOR(S) Meadows, C. A.			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Research Laboratory Washington, DC 20375-5000		8. PERFORMING ORGANIZATION REPORT NUMBER NRL-Report 9265	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Office of Naval Research Arlington, VA 22217-5000		10. SPONSORING / MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution unlimited.		12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) In this report we develop methods for analyzing key management and authentication protocols by using techniques developed for the solution of equations in a term-rewriting system. In particular, we describe a model of or class of protocols and possible attacks on those protocols as term-rewriting systems. We also describe a software tool based on the 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 various security properties. Two security flaws were found. A corrected scheme was also formally modeled and verified by using these techniques.			
14. SUBJECT TERMS Formal specification Verification Logic programming Protocols		15. NUMBER OF PAGES 38	16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL

CONTENTS

1. INTRODUCTION	1
2. DESCRIPTION OF THE MODEL AND PROOF TECHNIQUES USED	2
3. DESCRIPTION OF THE SELECTIVE BROADCAST PROTOCOL	6
4. MODELING THE SELECTIVE BROADCAST PROTOCOL	8
5. ANALYSIS OF THE SELECTIVE BROADCAST PROTOCOL	11
5.1 Obtaining Decrypted Words	12
5.2 Obtaining a Message in the Clear	15
5.3 Obtaining the Rights to an Encrypted Message	18
5.4 Substituting Messages	25
6. COMPARISON WITH OTHER WORK	30
7. DISCUSSION	31
8. CONCLUSIONS	33
9. ACKNOWLEDGMENTS	33
REFERENCES	33

APPLYING FORMAL METHODS TO THE ANALYSIS OF A KEY MANAGEMENT PROTOCOL

1. INTRODUCTION

It is difficult to be certain whether or not a cryptographic protocol satisfies its requirements. Protocols have been designed with subtle security flaws, independent of the strengths or weaknesses of the cryptoalgorithm used, that were not discovered until some time after they had been published. Examples include:

- the Needham-Schroeder key-distribution protocol [1] which was found to be vulnerable to various kinds of replay attacks [2,3].
- an early version of the IBM key management system which could be used by a penetrator to produce session keys in the clear [4], and
- the software protection scheme developed in Ref. 5, in which a penetrator could combine previously generated messages in such a way that the system could be induced to grant unauthorized access to software [6].

Other similar examples are discussed in Ref. 7. As we design more systems that depend on 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 Ref. 8, 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 gains greater assurance that the protocol satisfies its requirements. If it fails, the examination of the reasons for the failure may point out security flaws in the protocol.

Similar techniques have been applied in the design of communication protocols and of secure computing systems. In conclusion, it seems a likely conclusion that these techniques 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 report 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 report in Ref. 9, were successfully applied to the analysis of the selective broadcast protocol designed by G. J. Simmons [6]. The fact that the application of these techniques uncovered two security flaws in this protocol shows that they promise to be useful.

In Section 2 of this report we describe our formal model, which is an adaptation of Dolev and Yao's 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 3 we describe the selective broadcast protocol and the results of our analysis. In Section 4 we give the formal specification of the selective broadcast protocol. In Section 5 we describe the analysis of the specification. In Section 6 we compare our approach with other work on the specification and verification of cryptographic protocols. In Section 7 we discuss the implications of our results and the ways in which our system could be improved.

2. DESCRIPTION OF THE MODEL AND PROOF TECHNIQUES USED

The model that we use is an adaptation of the public-key model developed by Dolev and Yao in Ref. 10. We consider a protocol as a set of rules for passing messages among the participants. The protocol participant who receives a message, will, if he accepts it as genuine, generate a new message by performing certain operations on it or on other messages received earlier. Thus a cryptographic protocol may be thought of, in part, as a set of rules for generating words in a 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, 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 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 our model protocol rules are expressed as statements of the form

IF $I \subset W$ AND A AND $\text{Cond}(I,A)$ THEN $W := W \cup O$ AND A'

where 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 clauses consisting of conjunctions of clauses of the form $S = T$, where S is a state variable name, and $\text{Cond}(I,A)$ is a set of further conditions on I and A . Rules may be either deterministic or nondeterministic; that is, a rule may either always apply or only apply under circumstances that have not yet been specified. Rules of this form may be used to describe events as various as:

- (a) a participant in the protocol responding to a message from the penetrator,
- (b) a change in the internal state values of a participant,
- (c) a participant initiating an instance of a protocol, or
- (d) 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

IF $\{X,Y\} \subset W$ THEN $W := W \cup \{e(X,Y)\}$

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

IF $\{Y\} \subset W$ AND $\text{KEYSTATE}(a) = X$ THEN $W := W \cup \{e(X,Y)\}$.

As in Dolev and Yao, the security of the protocols we look at will depend on the inability of the penetrator to use the rules of the protocol to generate words that can be reduced by using a set of reduction rules to some word that the penetrator should not be able to learn. As we have argued in Ref. 9, the security of many protocols can be expressed at least partially, if not always entirely, in terms of such properties. Examples of such rules are:

- a. $d(X,e(X,Y)) \rightarrow Y$ 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
- b. $p(s(X)) \rightarrow X$ 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 2.1 — A specification of a protocol is a tuple

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

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 X and F , \hat{E} is a set of reduction rules defined on $T(\Phi,F)$, S is a set of state variable names, R is a set of protocol rules of the form

IF $I \subset W$ AND A AND $\text{Cond}(I,A)$ THEN $W := W \cup O$ AND A' ,

where I and O consist of terms from $T(\Phi,F)$ and A and A' are clauses made up of conjunctions of clauses of the form $T = D$, where T is a state variable name and $D \in T(\Phi,F)$, W_0 is set of words from $T(\Phi,F)$ initially known to the penetrator, and A_0 is a set of initial predicates on the state variables.

Definitions of terms and related ideas are given below.

Definition 2.2 — Let Φ be a countable set of variables, and let S be 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 $T(\Phi,S)$ denote the set of all terms made up from X and S , let $V(T)$ denote the set of variables in a term T , and let $F(T)$ denote the set of function symbols in T .

We will follow the conventions of Prolog and represent function symbols by lowercase words and variables by capital letters or words beginning with capital letters.

Definition 2.3 — A term-rewriting system over $T(\Phi,S)$ is a set of directed equations \hat{E} such that, for $T_1 \rightarrow T_2$ in \hat{E} , $V(T_2) \subset V(T_1)$. $\rightarrow_{\hat{E}}$ is the finest relation over $T(\Phi,S)$ containing \hat{E} and closed by substitution and replacement. We let $\rightarrow_{\hat{E}}^*$ (or simply, \rightarrow^*), denote the transitive-reflexive closure of $\rightarrow_{\hat{E}}$.

We will be interested in term-rewriting systems that are *noetherian* and *confluent*. That is, if term A is reachable from B in a sequence of term rewritings, then it is reachable in a finite number of such rewritings and if A is reducible to B and to C by two different single rewritings, then B and C are both reducible to a fourth term D. These systems have the property that every term A is reducible to a unique irreducible term $\text{nf}(A)$. See Ref. 11 for a discussion.

We can apply the rules of a protocol to move from one state to another. Suppose that we are in a state where a penetrator knows a set of terms L, and the statement $(S_1 = B_1, \dots, S_k = B_k)$ holds. We assume that all terms of L and all B_1 through B_k are irreducible, that is, that no further reduction rules can be applied. Thus, for example, if we assume that a penetrator knows a word $e(k, X)$, where k is a constant and X is a variable, this means that the penetrator knows some word $e(k, \sigma X)$ where σ is a substitution assigning X to a term such that $e(k, \sigma X)$ 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.* We then look for a rule $R =$

IF $I \subset W$ AND A AND $\text{Cond}(I, A)$ THEN $W := W \cup O$ AND A' ,

such that there is a substitution σ assigning terms to the variables of R, L, and B_1 through B_k such that $\sigma I \subset \sigma L$, $\sigma(S_1 = B_1, \dots, S_k = B_k)$ implies σA , the statement $\text{Cond}(\sigma I, \sigma A)$ is satisfied, and σL and σB_1 through σB_k remain irreducible. If this is the case, we know that the system state satisfies the conditions of rule R, and then we can move to the new system state. This is done by replacing L with $\sigma L \cup \text{nf}(\sigma O)$, where $\text{nf}(\sigma O)$ (or the normal form of σO) is the set of words obtained by applying all possible reductions to the words of $\sigma(O)$, and replacing $S_i = \sigma B_i$ with $S_i = \sigma B'_i$ if the former appears in $\sigma(S_1 = B_1, \dots, S_k = B_k)$ and the latter appears in $\text{nf}(\sigma A')$, and adding $T = \sigma C$ to $\sigma(S_1 = B_1, \dots, S_k = B_k)$ if $T = \sigma C$ appears in $\text{nf}(\sigma A')$ but not in $\sigma(S_1 = B_1, \dots, S_k = B_k)$.

For example, suppose the penetrator knows the word "message" and $\text{KEYSTATE}(a) = \text{key1}$. If we let σ be the substitution $Y := \text{message}$, $Z := a$, and $X := \text{key1}$, then we can apply the rule

IF $\{Y\} \subset W$ AND $\text{KEYSTATE}(Z) = X$ THEN $W := W \cup \{e(X, Y)\}$.

so that the new state becomes one in which the penetrator knows $\{\text{message}, e(\text{key1}, \text{message})\}$ and $\text{KEYSTATE}(a) = \text{key1}$.

To prove that a protocol is secure, we want to show that certain states cannot be reached from any initial state. 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 matching up a state description with the conclusion of a rule. But the techniques we apply are somewhat different than the techniques we apply to find what states can immediately follow a state. Suppose that, as before, we are in a state where a penetrator knows a set of words W, and the statement $(S_1 = B_1, \dots, S_k = B_k)$ holds; we now look for a substitution σ such that σO reduces to a subset of σL and $\sigma A'$ reduces to a statement that does not contradict $\sigma(S_1 = B_1, \dots, S_k = B_k)$. We then examine σO and σA and check that

- (a) σI , σA , σL , and $\sigma(S_1 = B_1, \dots, S_k = B_k)$ are irreducible
- (b) $\text{Cond}(\sigma I, \sigma A)$ is satisfied, and

*Our reasons for making this assumption have nothing to do with the mathematical basis of the term-rewriting program that we use, which could be applied equally as well if this restriction were not enforced. However, we have found this assumption useful in the analysis of protocols.

- (c) if $\sigma(S_1 = B_1, \dots, S_k = B_k)$ implies $\sigma(S_i = B_i)$ and σA contradicts $\sigma(S_i = B_i)$, then $\text{nf}(\sigma A')$ implies $\sigma(S_i = B_i)$; otherwise, $\text{nf}(\sigma A')$ fails to contradict $\sigma(S_i = B_i)$.

The description of the state immediately preceding is then one in which the penetrator knows $\sigma I \cup \sigma L$, and σA holds together with all $T = C$ from $\sigma(S_1 = B_1, \dots, S_k = B_k)$ that are not specified by σA .

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

IF $\{Y\} \subset W$ AND $\text{KEYSTATE}(Z) = X$ THEN $W := W \cup \{e(X, Y)\}$

to get the previous state to be one in which the penetrator knows Y and $\text{KEYSTATE}(Z) = X$. On the other hand, if we apply rule R and 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 $\text{KEYSTATE}(Z) = X$.

As we see from above, often there is more than one σ matching up 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. Usually, an infinite number of such σ exist, but it is often possible to find a finite and *complete* set of such σ , that is a set Σ such that, if τ 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 .

We can find such complete finite sets by using algorithms developed for finding complete sets of unifiers with respect to some equational theory [10]. The term-rewriting system we use is noetherian and confluent, thus we were able to use one of a class of algorithms, called *narrowing algorithms*, first discussed by Slagle [12], that can be used to generate complete sets of unifiers with respect to noetherian confluent term-rewriting systems. Accordingly we have written a program in Prolog [13] that uses a modified version of the NARROWER algorithm of Rety et al. [14] 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 .

Finally, once we have uncovered a mistake in a protocol, we would like to correct it and reverify that it satisfies the security properties in which we are interested. Our task will be easier if we do not have to reverify that the corrected protocol satisfies the security properties that the original protocol satisfied. To make this possible, we introduce the notion of a *restriction* of a rule.

Definition 2.4 — A rule $R_2 =$

IF $I_2 \subset W$ AND A_2 AND $\text{Cond}(I_2, A_2)$ THEN
 $W := W \cup O_2$ AND A_2'

is a restriction of a rule $R_1 =$

IF $I_1 \subset W$ AND A_1 AND $\text{Cond}(I_1, A_1)$ THEN
 $W := W \cup O_1$ AND A_1'

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

- (a) $\sigma I_1 \subset \sigma I_2$,
- (b) $\sigma A_2 \rightarrow \sigma A_1$,
- (c) $\sigma \text{Cond}(I_2, A_2) \rightarrow \sigma \text{Cond}(I_1, A_1)$,
- (d) $\sigma O_2 \subset \sigma O_1$, and
- (e) $\sigma A_1' = \sigma A_2'$.

It is easy to see that, if R_2 is a restriction of R_1 , then, if P is reachable from P' through an application of R_2 , then there is a state P'' that differs from P only in that the penetrator knows more words in P'' than in P , so that P'' is reachable from P' through the application of R_1 . Thus, if we specify a state by listing a set of terms that must be known to the penetrator and a list of values that must be assigned to a subset of that state variables, and no state satisfying that specification is reachable given a protocol specification containing a rule R , then no such state is reachable if we replace R by one or more restrictions of R .

3. DESCRIPTION OF THE SELECTIVE BROADCAST PROTOCOL

The selective broadcast protocol described in Ref. 6 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 that participant with the ability to decrypt a message cannot pass on the ability or the message 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 are stored in the processors when decrypted and are not made directly available to the participants. Examples of applications of this protocol are software protection, in which the messages would be software that would only be executed inside the tamper-proof processors, and key distribution, in which the messages would be session keys that would never appear in the clear outside the processors.

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 check the input data much; therefore they 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. Indeed, it was a flaw of this sort in an earlier similar protocol [5] that led to the development of the selective broadcast protocol.

The protocol is as follows. Each participant has access to a tamper-proof processor A that has access to a unique secret key $q(A)$. The processors also have access to a common key m . Both m 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, $e(m, q(A))$, 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 $d(q(A), d(m, J))$. He then sends the encrypted message and an encrypted key $e(m, e(q(B), d(q(A), e(q(B), d(q(A), d(m, J))))))$ to B . B inserts key and message, along with the identifier $e(m, q(A))$, into his processor. The processor uses its knowledge of m , $e(m, q(A))$, and $q(B)$ to decrypt the key; it uses the key to decrypt the message and then runs the message.

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

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

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

In the decrypt message mode, user B inputs $X = e(m,q(A))$, $Y = e(m,e(q(B),d(q(A),e(q(B),d(q(A),d(m,J))))))$ and the encrypted message. In this case the second output reduces to J . This is used to check that the key transfer has taken place correctly. The processor also decrypts the key and uses it to decrypt the message. 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 by 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 [5], suggested that a formal analysis was necessary. These facts made the protocol an excellent test case for our methods.

Our analysis found two security flaws in the protocol. Both were easily correctable, but they were subtle and might not have been found in a less formal analysis.

The first flaw, described in more detail in Section 5.3, was not found directly as the result of our attempts to prove the protocol correct, but it was found after inspection of some of the cases generated by the narrowing program. It relies on the incorrect operation of a penetrator's processor when it is used in decrypt message mode. If the processor operates correctly otherwise, it successfully detects the error, but the penetrator can use the diagnostic (that is, the second word output) generated by the processor to obtain a key to a message that he was not authorized to have. This flaw can be repaired by including less information in the diagnostic.

The reason that our methods did not directly lead to the discovery of this flaw is that our specification only described a system that operated correctly and did not take possible failures into account. In Section 6 we discuss how our system could be changed to assist us in finding these kinds of errors.

The other flaw, described in Section 5.4, can be exploited even if the system operates correctly in every respect, and it was found as a direct result of our use of the methods described in this report. By supplying incorrect information to his processor when it is run in encrypt message mode, a penetrator can encrypt his own message by 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 three security properties listed above.

One more interesting fact emerged in our analysis of the selective broadcast protocol. Nowhere in our proof of correctness did we need to use the fact that the processor identifiers $q(A)$ were unobtainable by a penetrator. This does not necessarily mean that the encryption of the values $q(A)$ was not needed; in some applications the system might have security needs beyond the ones that we analyzed in which the encryption of the processor identifiers was necessary. Moreover, the encryption of the processor identifiers may have some other effects necessary to system security besides keeping the identifiers secret. Nevertheless, the fact that we did not need to use the secrecy of the processor identifiers suggests that in certain applications we might be able to omit this layer of encryption, and that it would be worth our while to attempt to model and verify, with respect to the needs of a particular application, a system in which processor identifiers were unencrypted. This shows that our techniques can be used not only to locate security flaws of a system but to identify parts of a system that may be unnecessary to the achievement of security.

4. MODELING THE SELECTIVE BROADCAST PROTOCOL

We will assume that the system consists of an unspecified number of users, and that each user is represented by a processor T that is capable of generating an infinite number of messages $\text{mess}(T,N)$ to be encrypted, where N ranges from 1 to infinity. 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 will identify each processor under control of the penetrator by $q(\text{ID},\text{slave})$ where ID is a processor identifier, one of id_1 through id_k , and each processor not under control of the penetrator by $q(\text{ID},\text{free})$.

Since the protocol is meant to be used for several possible applications, the specification in Ref. 6 is necessarily incomplete. In particular, it is not specified how it is assured that each processor is accessible only by its owner, how a processor identifies the author of a request for an encrypted message, what other actions a processor takes upon receipt of a request besides generating the encrypted key (for example, if the message is a program, the owner of the program may want to charge the requester for its use), or what actions a processor takes after decrypting an encrypted message. All of these actions are dependent upon the application, and they would be specified once the application is chosen. Since they are also necessary to the security of the application, the protocol would have to be reverified once application-specific rules were inserted. However, verifying the specification before application-specific rules are inserted allows us to remove errors that would affect all applications, thus the verification process is easier later on.

We will give each processor five state variables. Each state variable takes on a term or a list of terms as a value. One, $\text{MESSTATE}(T)$, will hold the identifier of the next message that T can generate. Initially, $\text{MESSTATE}(T)$ is 1 (or $s(\text{zero})$, where s denoted the successor operator). The second, $\text{IDSTATE}(T)$, holds the identifier of a message to be decrypted by the user and the identifier of the originator of the message. The third, $\text{KEYSTATE}(T)$, holds the value of the decrypted key. The fourth, $\text{READYSTATE}(T)$, is set equal to "true" if the $\text{IDSTATE}(T)$ and $\text{KEYSTATE}(T)$ check out. The fifth, $\text{RUNSTATE}(T)$, 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
free	zero
slave	zero
name	two
identical	two
id1, ..., idk	zero
mess	two
m	zero

We use the following three rewrite rules:

1. $e(X, d(X, Y)) \rightarrow Y$
- $d(X, e(X, Y)) \rightarrow Y$
- $\text{identical}(X, X) \rightarrow \text{yes}.$

The proof that the term-rewriting system thus defined is noetherian and confluent is also similar to that given in Ref. 9, therefore we omit it here.

We first model the behavior of the free processors, which we assume behave according to the rules of the protocol. Each free processor can generate and encrypt messages. Each message is identified by a name denoted by $\text{name}(q(X, Y), N)$ 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 word $\text{name}(q(X, Y), N)$ does not necessarily uniquely identify the message, but the name together with $e(m, q(X, Y))$ does. The message itself (as opposed to its name) is denoted by $\text{mess}(q(X, Y), N)$. 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 $\text{MESSTATE}(q(\text{ID}, \text{free})) = N$ THEN
 $W := W \cup \{\text{name}(q(\text{ID}, \text{free}), N), e(d(q(\text{ID}, \text{free}), d(m, \text{name}(q(\text{ID}, \text{free}), J))), \text{mess}(q(\text{ID}, \text{free}), N))\}$
 AND $\text{MESSTATE}(q(\text{ID}, \text{free})) := s(J)$

where $s(N)$ denotes the successor of N .

Each free 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 $\text{name}(q(\text{ID}, \text{free}), N)$ 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 it 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 $\{X, Y\} \subset W$ THEN
 $W := W \cup \{e(m, q(ID, free)), e(m, e(d(m, X), d(q(ID, free), e(d(m, X), d(q(ID, free), d(m, Y))))))\}$.

Finally, a processor can attempt to decrypt a message. 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, zero], and that KEYSTATE, RUNSTATE, and READYSTA TE are zero. It also checks that the new identifiers are not zero. (This is in order that a hostile penetrator cause a state variable to be zeroed out.)

IF $\{X, Y\} \subset W$ AND IDSTATE(q(ID, free)) = [zero, zero] AND
 KEYSTATE(q(ID, free)) = zero AND READYSTA TE(q(ID, free)) = zero
 AND RUNSTATE(q(ID, free)) = zero
 AND $X \neq \text{zero}$ AND $nf(d(m, Y)) \neq \text{zero}$ THEN
 IDSTATE(q(ID, free)) := $\{X, d(m, Y)\}$.

Next, the processor accepts and decrypts a key. Again, it checks that KEYSTATE, READYSTA TE, and RUNSTATE do not hold any old key values, that the new key is not zero, and that the values stored in IDSTATE are not zero.

IF $\{X\} \subset W$ AND IDSTATE(q(ID, free)) = $\{Y, Z\}$
 AND KEYSTATE(q(ID, free)) = zero AND RUNSTATE(q(ID, free)) = zero
 AND READYSTA TE(q(ID, free)) = zero AND
 $Y \neq \text{zero}$ AND $Z \neq \text{zero}$
 AND $nf(d(q(ID, free), e(Z, d(q(ID, free), d(m, X)))))) \neq \text{zero}$ THEN
 KEYSTATE(q(ID, free)) :=
 $\{d(q(ID, free), e(Z, d(q(ID, free), d(m, X))))\}$.

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

IF IDSTATE(q(ID, free)) = $\{X, Y\}$ AND KEYSTATE(q(ID, free)) = $\{Z\}$
 AND READYSTA TE(q(ID, free)) = zero AND RUNSTATE(q(ID, free)) = zero
 AND $X \neq \text{zero}$ AND $Y \neq \text{zero}$ AND $Z \neq \text{zero}$ THEN
 READYSTA TE := $\{\text{identical}(X, e(m, e(Y, Z)))\}$.

The processor now decrypts the encrypted message. In doing so, it first checks that no message is presently stored in RUNSTATE, and that the new message is not zero.

IF $\{X\} \subset W$ AND KEYSTATE(q(ID, free)) = $\{Y\}$
 AND READYSTA TE(q(ID, free)) = [yes] AND RUNSTATE(q(ID, free)) = zero
 AND $Y \neq \text{zero}$ AND $nf(d(Y, X)) \neq \text{zero}$ THEN
 RUNSTATE(q(ID, free)) := $\{d(Y, X)\}$.

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.

IDSTATE(q(ID, free)) = zero AND KEYSTATE(q(ID, free)) = zero
 AND READYSTA TE(q(ID, free)) = zero
 AND RUNSTATE(q(ID, free)) = zero.

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

```
IF{ X, Y, Z} ⊂ W THEN
  W := W ∪ {e(m, q(ID, slave)),
    e(m, e(d(m, Y), d(q(ID, slave), e(d(m, Y), d(q(ID, slave), d(m, X)))))),
    e(d(q(ID, slave), e(d(m, Y), e(d(q(ID, slave), d(m, 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 to attempt to decrypt a message. These rules are identical to the rules for the free processors, except that in the rule describing READystate, the penetrator can learn the terminal output. The READystate rule is given as

```
IF IDSTATE(q(ID, slave)) = [X, Y] AND KEYSTATE(q(ID, slave)) = [Z]
  AND READystate(q(ID, slave)) = zero THEN
  READystate(q(ID, slave)) := [equal(X, e(m, e(Y, Z)))] AND
  W := W ∪ {e(m, q(ID, slave)), e(m, e(Y, Z))}.
```

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

```
IF{X, Y} ⊂ W THEN W := W ∪ {e(X, Y)}
IF{X, Y} ⊂ W THEN W := W ∪ {d(X, Y)}.
```

A penetrator may try to attain at least three goals. One is to decrypt a message on one of the processors under his control and that is not authorized to decrypt that message. 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 a free processor. Last but not least, the penetrator may try to obtain a copy of a free terminal's message in the clear.

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

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

5. ANALYSIS OF THE SELECTIVE BROADCAST PROTOCOL

To perform our analysis, we used the following definition and lemma from Ref. 9.

Definition 5.1 — Let E be a term in $T(\Phi, F)$. Let a be a function symbol of arity zero. We say that a appears on the *right-hand side* of E if $E = a$ or E is the result of applying a finite number of encryptions and decryptions to a .

Lemma 5.2 — Let $[T(\Phi, F), \hat{E}, S, R, W_0, A_0]$ be a specification of a cryptographic protocol. Let a be a function symbol of arity zero. Suppose that a does not appear on the right-hand side of any word in W_0 . Suppose also that, for every rule in R of the form

IF $S = T$ AND $I \subset W$ AND $\text{pred}(I, T)$ THEN
 $W := W \cup \{O_1, \dots, O_k\}$ AND $S := T'$.

a appears on the right-hand side of no word O_i , and if X is a variable appearing on the right-hand side of some O_i , then X appears on the right-hand side of some word in I . Then a is unobtainable, and any word in which a appears on the right-hand side is unobtainable.

Corollary 5.3 — The word m and any word with m on the right-hand side are unobtainable in the selective broadcast protocol.

Proof: All rules of the selective broadcast specification satisfy the hypotheses of Lemma 5.2 except the penetrator READYSTA TE rule. Thus the result is true for the protocol minus the penetrator READYSTA TE rule. Moreover, if H is a history in which the penetrator learns the word m or any word with m on the right-hand side in the last state change, that last change must have taken place via the penetrator READYSTA TE rule. Thus the penetrator learns a word $e(m, q(\text{ID}, \text{slave}), e(m, e(Y, Z)))$ reducible to a word with m on the right-hand side, and Z is the value of $\text{KEYSTATE}(q(\text{ID}, \text{slave}))$. By inspection of the reduction rules we see that Z must have m on the right-hand side. Application of the narrow program shows us that, for $\text{KEYSTATE}(q(\text{ID}, \text{slave}))$ to take on the value Z , we must have $Z = d(q(\text{ID}, \text{slave}), e(Q, d(q(\text{ID}, \text{slave}), d(m, X))))$, with previous knowledge of X . But X must also have m on the right-hand side, which contradicts our assumption that Z is the first word learned with m on the right-hand side. \square

We are now ready to begin the analysis.

5.1 Obtaining Decrypted Words

In our initial attempts to run the narrower program to answer these questions, we kept on running up against the following question: Is it possible for the penetrator to obtain a word $d(X, Y)$ without first knowing X and Y . We decided to try to answer that question first.

We assumed that we were at a point in the history in which previously, if an irreducible instance of $d(X, Y)$ had been obtained, then X and Y had been obtained previously to $d(X, Y)$. In the first run of the narrower program, we asked the program for all cases in which the penetrator could find $d(X, Y)$ without assuming previous knowledge of m , both X and Y , $d(X, Y)$, or $d(Z, d(X, Y))$. (We included the last because by our hypothesis knowledge of $d(Z, d(X, Y))$ would imply knowledge of Z and $d(X, Y)$.)

There were a number of responses. In each, either knowledge of an irreducible instance of $e(m, e(q(W), d(Z, e(q(W), d(Z, d(m, d(X, Y)))))))$ or knowledge of both Z and $e(Z, d(X, Y))$ was required. After asking the narrowing program how these words could be obtained, it was decided to try to show the following.

Lemma 5.4 — Assume that, at a particular point in the protocol, no irreducible instance of $d(W_1, W_2)$ has been obtained where one of W_1 and W_2 have not been previously computed. Let X and Y be two words, one of which has not been previously computed, and assume that in the next

step of the protocol an irreducible word of F is obtained, where F is the language defined by the following productions:

$$F: \rightarrow d(X,Y)$$

$$F: \rightarrow e(Q,F)$$

$$C: \rightarrow d(m,F)$$

$$C: \rightarrow e(L,d(L,C))$$

$$F: \rightarrow e(m,e(L,d(L,C)))$$

where L is the set of all irreducible terms, and Q is the set of all irreducible terms except m . Then some word of F must have been obtained prior to this point.

Since no word of F is known to the penetrator initially, this means that all words of F are unobtainable, and that in particular $d(X,Y)$ is unobtainable.

Proof: Running the narrower on $d(X,Y)$ showed us that learning $d(X,Y)$ required knowledge of a word in F . Thus, to show that knowledge of a word in F requires previous knowledge of a word in F we need to show that

- (1) knowledge of a word $e(Q,F)$ requires previous knowledge of a word in F or of an unobtainable word, and
- (2) knowledge of a word from $e(m,e(L,d(L,C)))$ requires previous knowledge of a word from F or of an unobtainable word.

To answer (1), the next question we posed to the narrowing program was: how can we find instances of $e(Z,W)$ where $Z \neq m$ and W was not previously known? This was done by asking the narrowing program to find $e(Z,W)$, but to throw out solutions of the form $e(m,W)$. The program was also told to throw out solutions that required input of m , W , or $d(R,e(Z,W))$, since m is unobtainable, W is assumed not to be previously known, and knowledge of $d(R,e(Z,W))$ would by hypothesis require previous knowledge of $e(Z,W)$. All solutions turned out to require previous knowledge instances of

- (a) $e(m,e(A,d(B,e(A,d(B,d(m,e(Z,W)))))))$, or;
- (b) $e(A,e(Z,W))$.

Assuming that W was in F , all words of the form listed above would also have to be in F . Thus (1) holds.

To show that (2) held, we asked the narrower program how to obtain instances of $e(m,e(A,d(B,C)))$. We assumed that $C \in C$. We asked the program to rule out solutions requiring previous knowledge of m , $e(m,C)$ (since $e(m,C)$ is assumed to lie in F if irreducible), and $d(Z,e(m,e(A,d(B,C))))$. Quite a few solutions were produced. All of them required, previous knowledge of instances of one of the following:

- (a) $e(m, e(D, d(E, e(D, d(E, e(A, d(B, C)))))))$
- (b) $e(m, e(D, d(E, e(D, d(B, C)))))$
- (c) $e(m, e(E, d(A, C)))$
- (d) $e(m, e(D, E))$ where $C = e(A, E)$
- (e) $e(m, E)$ where $C = e(A, d(D, E))$
- (f) E where $C = e(A, d(D, d(m, E)))$
- (g) $e(m, e(D, m))$
- (h) $e(m, e(D, d(E, e(D, m))))$
- (i) $e(D, e(m, e(A, d(B, C))))$ and D .

Some cases required no input words, but required that KEYSTATE and IDSTATE must have certain values. When we ran the narrower program on these, however, we found that they required prior knowledge of one of the words listed above.

We consider each case.

In case (a) — $e(m, e(A, d(B, C)))$ is in F by hypothesis. Hence so is $e(m, e(D, d(E, e(A, d(B, C)))))$ and thus so is $e(m, e(D, d(E, e(D, d(E, e(A, d(B, C)))))))$.

In case (b) — $C \in C$ by hypothesis, so $e(m, e(D, d(E, e(D, d(B, C)))) \in F$. A similar argument works for case (c).

In case (d) — the fact that $C \in C$ means that $C \in d(m, F)$ or $e(L, d(L, C))$. Since $C = e(A, E)$, C must be a member of the latter. Thus $E = d(R, S)$, where $S \in C$. Thus $e(m, e(D, E))$ is in F . A similar argument works for cases (e) and (f).

In cases (g) and (h), the word m appears on the right-hand side of the input and so those words are unobtainable.

In case (i) — D cannot be m , since m is unobtainable, and so $e(D, e(m, e(A, d(B, C))))$ must be in F .

We now show that $d(X, Y)$ is unobtainable unless the penetrator already knows X and Y . Suppose that a penetrator has obtained $d(X, Y)$ without having previously obtained X and Y . Then the penetrator must have already obtained a word from F . Let W be the first such word obtained by the penetrator. If W were $d(X, Y)$, then the penetrator would have already known a word from F . If W were in $e(Q, F)$, then by our analysis of the results of running the narrower on $e(A, B)$, the penetrator would have previously obtained a word from F . Finally, if W were in $e(m, e(L, d(L, C)))$, then by our analysis of the results of running the narrower on $e(m, e(A, d(B, C)))$, the penetrator would have already obtained a word from F . Since the penetrator does not know any word of F initially, we can show by induction that F , and hence $d(X, Y)$, is unobtainable. []

As a corollary we obtain the result that a penetrator cannot find the key to a message encrypted by a free terminal, for such keys are of the form $d(q(\text{ID}, \text{free}), d(m, \text{name}(q(\text{ID}, \text{free}), N)))$. Knowledge of such a key would imply knowledge of $q(\text{ID}, \text{free})$ and $d(m, \text{name}(q(\text{ID}, \text{free}), N))$, which would in turn imply knowledge of m . But m is unobtainable.

5.2 Obtaining a Message in the Clear

Our next goal was to find out whether or not a penetrator could obtain a message $\text{mess}(q(\text{ID}, \text{free}), N)$ in the clear. First we asked the narrower to find under what conditions a penetrator could obtain such a word. We asked it to ignore solutions that required the penetrator to have prior knowledge of m or $d(Z, \text{mess}(q(\text{ID}, \text{free}), N))$, since by Lemma 5.3 m is unobtainable, and by Lemma 5.4, $d(Z, \text{mess}(q(\text{ID}, \text{free}), N))$ is unobtainable unless $\text{mess}(q(\text{ID}, \text{free}), N)$ is already known. The program derived a number of solutions, each of which required one of the following conditions:

- (a) $\text{KEYSTATE}(q(Y, \text{slave})) = d(Z, d(m, \text{mess}(q(\text{ID}, \text{free}), N))$ and $\text{IDSTATE}(q(Y, \text{slave})) = [R, Z]$
- (b) prior knowledge of $e(m, e(X, d(Z, e(X, d(Z, d(m, \text{mess}(q(\text{ID}, \text{free}), N))))))$
- (c) prior knowledge of X and $e(X, \text{mess}(q(\text{ID}, \text{free}), N))$.

When we ran the program on the first condition, we found that attaining that state required prior knowledge of one of the following:

- (a) $\text{mess}(q(\text{ID}, \text{free}), N)$
- (b) $e(m, e(q(\text{ID}2, \text{slave}), d(Y, d(m, \text{mess}(q(\text{ID}, \text{free}), N))))$
- (c) $e(m, e(q(\text{ID}2, \text{slave}), d(Y, e(q(\text{ID}2, \text{slave}), d(Z, d(m, \text{mess}(q(\text{ID}, \text{free}), N))))))$.

This and the results of running the narrower program on the last two conditions prompted us to try and verify the unobtainability of the irreducible words of the following language **A**:

- A**: $\rightarrow \text{mess}(q(\text{ID}, \text{free}), N)$
- A**: $\rightarrow e(\mathbf{B}, \mathbf{A})$
- C**: $\rightarrow d(m, \mathbf{A})$
- C**: $\rightarrow e(\mathbf{L}, d(\mathbf{L}, \mathbf{C}))$
- C**: $\rightarrow e(\mathbf{L}, d(\mathbf{L}, \mathbf{A}))$
- A**: $\rightarrow e(m, \mathbf{C})$

where **L** is the set of all irreducible terms and **B** is the set of all irreducible terms except $d(q(\text{ID}, \text{free}), d(m, \text{name}(q(\text{ID}, \text{free}), N)))$.

Theorem 5.5 — **A** as defined above is unobtainable.

Proof: The results of running the narrowing program on $\text{mess}(q(\text{ID}, \text{free}), N)$ show us that knowledge of $\text{mess}(q(\text{ID}, \text{free}), N)$ already implies prior knowledge of a word of A . Thus we can show that A is unobtainable if we show that

- (1) knowledge of $e(A, B)$ where $A \neq d(q(\text{ID}, \text{free}), d(m, \text{name}(q(\text{ID}, \text{free}), N)))$ and $B \in A$ implies previous knowledge of a word from A ;
- (2) knowledge of $e(m, e(A, d(B, C)))$ where $C \in C$ implies previous knowledge of a word from A , and;
- (3) knowledge of $e(m, e(A, d(B, C)))$ where $C \in A$ implies previous knowledge of a word from A .

To show (1), we run the narrower on $e(A, B)$ again, but this time we do not exclude instances of $e(m, B)$, but instances of $e(d(q(\text{ID}, \text{free}), d(m, \text{name}(q(\text{ID}, \text{free}), N))), \text{mess}(q(\text{ID}, \text{free}), N))$. We also exclude cases that require previous knowledge of m or $d(C, e(A, B))$. We came up with cases that required previous knowledge of one of the following:

- (a) $e(m, e(q(S, T), d(V, e(q(S, T), d(V, d(m, e(A, B)))))))$
- (b) $e(m, e(q(S, T), d(V, e(q(S, T), d(V, B)))))$
- (c) $e(m, e(q(S, T), d(V, e(q(S, T), W))))$ where $B = e(V, W)$ and $e(m, V)$
- (d) $e(m, e(q(S, T), d(V, W)))$ where $B = e(V, d(q(S, T), W))$ and $e(m, V)$
- (e) $e(m, e(q(S, T), W))$ and $e(m, V)$ where $B = e(V, d(q(S, T), e(V, W)))$
- (f) $e(m, W)$ and $e(m, V)$ where $B = e(V, d(q(S, T), e(V, d(q(S, T), W))))$
- (g) W and $e(m, V)$ where $B = e(V, d(q(S, T), e(V, d(q(S, T), d(m, W)))))$
- (h) $e(m, e(q(S, T), d(m, V), e(q(S, T), W)))$ and V where $B = e(d(m, V), W)$
- (i) $e(m, e(q(S, T), d(d(m, V), W)))$ and V where $B = e(d(m, V), d(q(S, T), W))$
- (j) $e(m, e(q(S, T), W))$ and V where $B = e(d(m, V), d(q(S, T), e(d(m, V), W)))$
- (k) $e(m, W)$ and V where $B = e(d(m, V), d(q(S, T), e(d(m, V), d(q(S, T), W))))$
- (l) W and V where $B = e(d(m, V), d(q(S, T), e(d(m, V), d(q(S, T), d(m, W)))))$
- (m) B
- (n) $e(C, e(A, B))$ and C

We want to show that knowledge of $e(A, B)$ implies previous knowledge of a word from A .

Cases (a) and (b) can be easily shown to imply previous knowledge of a word from A .

In case (c) — $B = e(V, W)$. Since $B \in A$, either W is in A or $V = m$ and $W \in C$. But $e(m, V)$ is known, so V cannot be m . Thus $W \in A$. Hence $e(q(S, T), W) \in A$, and so $e(m, e(q(S, T), d(V, e(q(S, T), W)))) \in A$. A similar argument works for case (h).

In case (d) — $B = e(V, d(q(S, T), W))$, and so $d(q(S, T), W)$ must be in A . But A contains no elements of this form. A similar argument works for cases (e) through (g) and (i) through (l).

In case (m) — previous knowledge of $B \in A$ is needed.

In case (n) — previous knowledge of $e(C, e(A, B))$ and C is needed. Since C is known, C cannot be $e(q(ID, free), d(m, N))$, so $e(A, e(A, B)) \in A$.

Next we consider case (2). We take the output of the narrower on $e(m, e(A, d(B, C)))$ in which it was asked to ignore solutions requiring previous knowledge of $e(m, C)$ (in A), m , or $d(Z, e(m, e(A, d(B, C))))$. Recall that these required previous knowledge of one of the following:

- (a) $e(m, e(Y, d(X, e(Y, d(X, e(A, d(B, C)))))))$
- (b) $e(m, e(Y, d(X, e(Y, d(B, C))))))$
- (c) $e(m, e(Y, d(A, C)))$
- (d) $e(m, e(Y, X))$ where $C = e(A, X)$
- (e) $e(m, X)$ where $C = e(A, d(Y, X))$
- (f) X where $C = e(A, d(Y, d(m, X)))$
- (g) $e(m, e(Y, m))$
- (h) $e(m, e(Y, d(X, e(Y, m))))$
- (i) $e(X, e(m, e(A, d(B, C))))$ and X .

The same previous arguments also hold in this case, except for case (1). For that case, we note that, since X is known, it cannot be $d(q(ID, free), d(m, name(q(ID, free), N)))$, therefore $e(X, e(m, e(A, d(B, C)))) \in A$.

Finally, we consider case (3). We run the narrower on $e(m, e(A, d(B, C)))$ where C is now presumed to be in A , this time asking it to ignore all results that require previous knowledge of C , $d(Z, C)$, m , or $d(Z, e(m, e(A, d(B, C))))$. (Words of the form $d(Z, C)$ are excluded because knowledge of such a word would imply previous knowledge of C). We found that previous knowledge of one of the following was required:

- (a) $e(m, e(q(R, S), d(X, e(q(R, S), d(X, e(A, d(B, C)))))))$
- (b) $e(m, e(q(R, S), d(X, e(q(R, S), d(B, C))))))$
- (c) $e(m, e(q(R, S), d(A, C)))$
- (d) $e(m, e(q(R, S), Y))$ and $e(m, A)$ where $C = e(A, Y)$

- (e) $e(m, Y)$ and $e(m, A)$ where $C = e(A, d(q(R, S), Y))$
- (f) Y and $e(m, A)$ where $C = e(A, d(q(R, S), d(m, Y)))$
- (g) $e(m, e(q(R, S), Y))$ and Z where $C = e(d(m, Z), Y)$
- (h) $e(m, Y)$ and Z where $C = e(d(m, Z), d(q(R, X), Y))$
- (i) Y and Z where $C = e(d(m, Z), d(q(R, X), d(m, Y)))$
- (j) words with m on the right-hand side
- (k) $e(X, e(m, e(A, d(B, C))))$ and X

Cases (a), (b), and (c) are in A by the fact that $C \in A$ and the definition of A .

In case (d), $e(m, A)$ is known, so $A \neq m$. Thus, since $C = e(A, Y)$ and $C \in A$, $Y \in A$.

In cases (e) and (f), $C \in A$ and $A \neq m$ as before, so $d(q(R, S), Y)$ and $d(q(R, S), d(m, Y))$ must be in A . But they fail to satisfy the definition.

In case (g), $d(m, Z) \neq m$, so $Y \in A$, and so $e(m, e(q(R, S), Y)) \in A$ by definition.

Cases (h) and (i) are similar to cases (e) and (f), with $d(m, Z)$ substituted for A .

In case (j), words with m on the right-hand side are unobtainable.

In case (k), the fact that X is known means that $X \neq d(q(ID, free), d(m, name(q(ID, free), N)))$, and thus $e(X, e(m, e(A, d(B, C)))) \in A$ by definition.

Thus we have shown that A , and hence $mess(q(ID, free), N)$, is unobtainable. []

As a corollary, we have that the only words of the form $e(Z, mess(q(X, free), Y))$ obtainable by the penetrator are those where Z is $d(q(X, free), d(m, name(q(X, free), Y)))$.

5.3 Obtaining the Rights to an Encrypted Message

Now we show that a penetrator cannot decrypt a message without first paying for it. In other words, we will prove the following theorem:

Theorem 5.6 — It is impossible for $RUNSTATE(q(ID, slave)) = mess(q(ID2, free), N)$ unless the rule

IF $\{X, J\} \subset W$ THEN

$W := W \cup \{e(m, q(ID, free)), e(m, e(d(m, X), d(q(ID, free), e(d(m, X), d(q(ID, free), d(m, J))))))\}$

has been applied with $X = e(m, q(ID, slave))$ and $J = name(q(ID, free), N)$.

To prove this theorem, first we asked the narrower program for the conditions necessary for the state value $RUNSTATE(q(ID, slave)) = mess(q(ID2, free), N)$. It responded that $KEYSTATE(q(ID, slave))$ must be X where the penetrator has prior knowledge of $e(X, mess(q(ID2, free), N))$, and that $READYSTATE(q(ID, slave))$ must be 'yes'.

According to the results of Section 5.2, X must be equal to $d(q(\text{ID2}, \text{free}), d(m, \text{name}(q(\text{ID2}, \text{free}), N)))$. Thus our next step was to ask the narrower program what were the conditions for $\text{KEYSTATE}(q(\text{ID}, \text{slave})) = d(q(\text{ID2}, \text{free}), d(m, \text{name}(q(\text{ID2}, \text{free}), N)))$ and $\text{READYSTATE}(q(\text{ID}, \text{slave})) = \text{'yes'}$. The narrower responded:

- (a) $\text{KEYSTATE}(q(\text{ID}, \text{slave})) = d(q(\text{ID2}, \text{free}), d(m, \text{name}(q(\text{ID}, \text{free}), N)))$ and $\text{IDSTATE}(q(\text{ID}, \text{slave})) = [\text{name}(q(\text{ID}, \text{free}), N), q(\text{ID2}, \text{free})]$, and;
- (b) $\text{KEYSTATE}(q(\text{ID}, \text{slave})) = d(q(\text{ID2}, \text{free}), d(m, \text{name}(q(\text{ID}, \text{free}), N)))$ and $\text{IDSTATE}(q(\text{ID}, \text{slave})) = [e(m, e(Y, d(q(\text{ID2}, \text{free}), d(m, \text{name}(q(\text{ID2}, \text{free}), N))))), Y]$.

We asked the narrower what conditions were necessary for (a) or (b) to hold. It responded that, in the first case, prior knowledge of

$$e(m, e(q(\text{ID}, \text{slave}), d(q(\text{ID2}, \text{free}), e(q(\text{ID}, \text{slave}), d(q(\text{ID2}, \text{free}), d(m, \text{name}(q(\text{ID2}, \text{free}), N)))))))$$

was necessary, and in the second, either prior knowledge of

$$e(m, e(q(\text{ID}, \text{slave}), d(Y, e(q(\text{ID}, \text{slave}), d(q(\text{ID2}, \text{free}), d(m, \text{name}(q(\text{ID2}, \text{free}), N)))))))$$

was necessary, or prior knowledge of

$$e(m, e(q(\text{ID}, \text{slave}), d(q(\text{ID2}, \text{free}), d(m, \text{name}(q(\text{ID2}, \text{free}), N))))))$$

and

$$\text{IDSTATE} = [e(m, e(q(\text{ID}, \text{slave}), d(q(\text{ID2}, \text{free}), d(m, \text{name}(q(\text{ID2}, \text{free}), N))))), q(\text{ID}, \text{slave})]$$

was necessary. Thus all we need to show is that none of the above words are obtainable without application of the rule

IF $\{X, J\} \subset W$ THEN

$$W := W \cup \{e(m, q(\text{ID}, \text{free})), e(m, e(d(m, X), d(q(\text{ID}, \text{free}), e(d(m, X), d(q(\text{ID}, \text{free}), d(m, J))))))\}$$

with $X = e(m, q(\text{ID}, \text{slave}))$ and $J = \text{name}(q(\text{ID}, \text{free}), N)$.

First we asked the narrowing program what conditions would be needed for the penetrator to learn

$$e(m, e(q(\text{ID}, \text{slave}), d(q(\text{ID2}, \text{free}), d(m, \text{name}(q(\text{ID2}, \text{free}), N))))),$$

asking it to ignore conditions that required previous knowledge of m ,

$$d(q(\text{ID2}, \text{free}), d(m, \text{name}(q(\text{ID2}, \text{free}), N))),$$

or

$$d(Z, e(m, e(q(\text{ID}, \text{slave}), d(q(\text{ID2}, \text{free}), d(m, \text{name}(q(\text{ID2}, \text{free}), N)))))).$$

It responded with a number of conditions, each of which involved at least one of the following requirements:

- (a) $\text{KEYSTATE}(q(B, \text{slave})) = d(q(\text{ID2}, \text{free}), d(m, \text{name}(q(\text{ID2}, \text{free}), N)))$
and $\text{IDSTATE} = [X, q(\text{ID}, \text{slave})]$ for some B;
- (b) $\text{KEYSTATE}(q(B, \text{slave})) = d(X, e(q(\text{ID}, \text{slave}), d(q(\text{ID2}, \text{free}), d(m, \text{name}(q(\text{ID2}, \text{free}), N))))))$
and $\text{IDSTATE} = [Y, X]$;
- (c) prior knowledge of
 $e(m, e(q(A, \text{free}), d(X, e(q(A, \text{free}), d(X, e(q(\text{ID}, \text{slave}), d(q(\text{ID2}, \text{free}), d(m, \text{name}(q(\text{ID2}, \text{free}), N))))))))))$;
- (d) prior knowledge of
 $e(m, e(q(A, \text{free}), e(q(\text{ID}, \text{slave}), e(q(A, \text{free}), d(q(\text{ID2}, \text{free}), d(m, \text{name}(q(\text{ID2}, \text{free}), N))))))))$;
- (e) prior knowledge of
 $e(m, e(q(\text{ID2}, \text{free}), d(q(B, \text{slave}), d(m, \text{name}(q(\text{ID2}, \text{free}), N))))))$;
- (f) prior knowledge of
 $e(m, e(q(B, \text{slave}), d(X, e(q(B, \text{slave}), d(X, e(q(\text{ID}, \text{slave}), d(q(\text{ID2}, \text{free}), d(m, \text{name}(q(\text{ID2}, \text{free}), N))))))))$;
- (g) prior knowledge of
 $e(m, e(q(B, \text{slave}), d(q(\text{ID}, \text{slave}), e(q(B, \text{slave}), d(q(\text{ID2}, \text{free}), d(m, \text{name}(q(\text{ID2}, \text{free}), N))))))))$;
- (h) prior knowledge of
 X and $e(X, e(m, e(q(\text{ID}, \text{slave}), d(q(\text{ID2}, \text{slave}), \text{name}(q(\text{ID2}, \text{free}), N))))))$, or;
- (i) prior knowledge of a word with m on the right-hand side.

Condition a raises some interesting questions. When we ran the narrower on the **KEYSTATE** value described in condition a, we found that to obtain it, the penetrator needs only to have prior knowledge of the word

$$e(m, e(q(B, \text{slave}), d(Q, e(q(B, \text{slave}), d(q(\text{ID2}, \text{free}), d(m, \text{name}(q(\text{ID2}, \text{free}), N))))))))),$$

and that $\text{IDSTATE}(q(B, \text{slave})) = [X, Q]$. If $Q = q(\text{ID2}, \text{free})$, then processor $q(B, \text{slave})$ could have obtained the word by buying $\text{mess}(q(\text{ID2}, \text{free}), N)$ from terminal $q(\text{ID2}, \text{free})$. Since we need $\text{IDSTATE}(q(B, \text{slave}))$ to be equal to $[X, q(\text{ID}, \text{slave})]$ for condition a to hold, then knowledge of

$$e(m, e(q(B, \text{slave}), d(q(\text{ID2}, \text{free}), e(q(B, \text{slave}), d(q(\text{ID2}, \text{free}), d(m, \text{name}(q(\text{ID2}, \text{free}), N))))))))$$

is not enough to bring about condition a. However, if it is possible for a penetrator to exploit some flaw in the system software or hardware to force $\text{IDSTATE}(q(B, \text{slave}))$ to take on the value $[X, q(\text{ID}, \text{slave})]$ when it should be $[X, q(\text{ID2}, \text{free})]$, the penetrator will be able to achieve condition a, obtain the word

$$(m, e(q(ID, slave), d(q(ID2, free), d(m, name(q(ID2, free), N))))))$$

and thus be able to have $mess(q(ID2, free), N)$ decrypted by processor $q(ID, slave)$ without authorization from terminal $q(ID2, free)$. Thus, we have found, if not exactly a flaw in the protocol, a point of vulnerability. The diagnostics that are used to detect and inform the user of a system flaw can cause a security flaw in the protocol.

However this vulnerability is easy to fix, assuming that we consider it a threat. Instead of having the processor output its attempt to decrypt the encrypted value of the message number in this case, we can require the user to insert the message number, have the processor do the decryption, compare the decrypted value with the message number, and output "yes" or "no."

Attaining either condition a or b requires prior knowledge of one of the following:

- (a) $e(m, e(q(ID, slave), d(q(ID2, free), d(m, name(q(ID2, free), N))))))$ with $B = ID$
- (b) $e(m, e(q(B, slave), d(q(ID, slave), e(q(B, slave), d(q(ID2, free), d(m, name(q(ID2, free), N)))))))$
- (c) $e(m, e(q(B, slave), d(X, e(q(B, slave), d(X, e(q(ID, slave), d(q(ID2, free), d(m, name(q(ID2, free), N))))))))))$.

After some experimentation we decided to show that the penetrator could not learn an *odd* where odd words are defined as follows.

To define odd words we need the standard definition of occurrence.

Definition 5.7 — If T is a term, we define the set of occurrences in T , $O(T)$ and their values as follows,

- (1) $\lambda \in O(T)$ and $T/\lambda = T$.
- (2) If $T = f(T_1, \dots, T_n)$, and $o \in O(T_i)$, then $i, o \in O(T)$ and $T/i.o = T_i/o$.

If $o \neq \lambda$, 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 .

We can now define odd words.

Definition 5.8 — We begin by defining even words. These are all words from $T(\Phi, F)$ containing no variable symbols that satisfy the following criteria:

- (a) The right-hand side of the word is $d(m, name(q(ID2, free), N))$.
- (b) There is at least one other occurrence of m at some $1.1 \dots 1.1$ such that $e(m, Y)$ occurs at the prefix of $1.1 \dots 1.1$.
- (c) Let O be the last such occurrence. Then, for all expressions E , the number of occurrences S of the form $O.1.1 \dots 1$ such that E occurs at S is even.

A word is *odd* if it satisfies the first two criteria and not the last.

A word from $T(\Phi, F)$ containing variable symbols is even (respectively, odd) if it satisfies those first two criteria and all instances are even (respectively, odd). Thus

$$e(m, e(q(A, \text{free}), d(q(B, \text{slave}), d(m, \text{name}(q(\text{ID2}, \text{free}), N))))))$$

is odd, and so is

$$e(q(B, \text{slave}), e(m, e(q(B, \text{slave}), d(m, \text{name}(q(\text{ID2}, \text{free}), N))))).$$

However,

$$e(m, e(X, d(Y, e(X, d(Y, d(m, \text{name}(q(\text{ID2}, \text{free}), N)))))))$$

is even. The word

$$e(m, e(X, e(Y, d(m, \text{name}(q(\text{ID2}, \text{free}), N))))))$$

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

Note that

$$e(m, e(q(\text{ID}, \text{slave}), d(q(\text{ID2}, \text{free}), d(m, \text{name}(q(\text{ID2}, \text{free}), N))))))$$

is itself an odd word, and that the results of the narrower show that previous knowledge of an odd word is required in order to learn it.

Lemma 5.9 — Odd words are unobtainable.

Proof: We ran the narrower on X , where X was assumed to be an odd word. All of the necessary conditions turned up by the narrower fall into one of the following cases:

Prior knowledge of one of the following:

- (a) $e(m, e(S, d(R, e(S, d(R, d(m, X))))))$
- (b) $e(m, e(S, d(R, e(S, d(R, C)))))$ where $X = e(m, C)$
- (c) $e(m, e(S, d(R, e(S, C))))$ where $X = e(m, e(R, C))$
- (d) $e(m, e(S, d(R, C)))$ where $X = e(m, e(R, d(S, C)))$
- (e) $e(m, e(S, C))$ where $X = e(m, e(R, d(S, e(R, C))))$
- (f) $e(m, C)$ where $X = e(m, e(R, d(S, e(R, d(S, C)))))$
- (g) C where $X = e(m, e(R, d(S, e(T, d(S, d(m, C))))))$
- (h) $e(m, e(q(A, B), D))$ and C where $X = e(D, C)$
- (i) C and $e(m, D)$ where $X = e(d(q(A, B), D), C)$

- (j) C and D where $X = e(d(q(A,B),d(m,D)),C)$
- (k) C and $e(m,e(q(A,B),d(R,e(q(A,B),D))))$ where $X = e(D,C)$
- (l) C and D where $X = e(d(q(A,B),e(R,d(q(A,B),d(m,D))))),C)$
- (m) C and $e(m,D)$ where $X = e(d(q(A,B),e(R,d(q(A,B),D))),C)$
- (n) C and $e(m,e(q(A,B),D))$ where $X = e(d(q(A,B),e(R,D)),C)$
- (o) $e(D,X)$ and D
- (p) C and D where $X = e(D,C)$
- (q) C and D where $X = d(D,C)$

The narrower also found the following state values that would produce an odd word

- (r) $KEYSTATE(q(A,slave)) = C$ and $IDSTATE(q(A,slave)) = [R,D]$ where $X = e(m,e(D,C))$
- (s) $KEYSTATE(q(A,slave)) = d(D,C)$ and $IDSTATE(q(A,slave)) = [R,D]$ where $X = e(m,C)$
- (t) $KEYSTATE(q(A,slave)) = d(D,d(m,X))$ and $IDSTATE(q(A,slave)) = [R,D]$.

Clearly, cases (a) through (g) require previous knowledge of an odd word. In cases (h), (k), (o), and (p) C is an odd word unless $D = m$, in which case $e(m,e(q(A,B),D))$, $e(m,e(q(A,B),d(R,e(q(A,B),D))))$, and D, respectively, are unobtainable. Cases (i), (j), (l) through (n), and (q) require previous knowledge of an odd word.

By running the narrower program on the values for KEYSTATE, we see that prior knowledge of one of the following is required.

For case (r)

- r1. X

For case (s)

- s1. X
- s2. $e(m,e(q(A,B),d(D,e(q(A,B),d(D,C)))))$ where $X = e(m,C)$.

For case (t)

- t1. X
- t2. $e(m,e(q(A,B),d(D,e(q(A,B),d(D,d(m,X))))))$

In each case, previous knowledge of an odd word was required. \square

Now recall that we are attempting to show that any irreducible instance of

$$e(m, e(q(ID, slave), d(q(ID2, free), e(q(ID, slave), d(q(ID2, free), d(m, name(q(ID2, free), N)))))))));$$

$$e(m, e(q(ID, slave), d(Y, e(q(ID, slave), d(q(ID2, free), d(m, name(q(ID2, free), N))))))), \text{ or};$$

$$e(m, e(q(ID, slave), d(q(ID2, free), d(m, name(q(ID2, free), N))))))$$

is unobtainable unless terminal $q(ID, slave)$ has obtained the rights to $mess(q(ID2, free), N)$ from terminal $q(ID2, free)$. However the first and third words are odd, and so is every instance of the second, except when $Y = q(ID2, free)$. Thus all that remains to be shown is that

$$e(m, e(q(ID, slave), d(q(ID2, free), e(q(ID, slave), d(q(ID2, free), d(m, name(q(ID2, free), N))))))))$$

is unobtainable unless terminal $q(ID, slave)$ has obtained the rights to $mess(q(ID2, free), N)$ from terminal $q(ID2, free)$. After running the narrower on that word, we found that all cases in which the penetrator was not required to know a word that was already shown to be unobtainable or to have purchased the program to run on processor $q(ID, slave)$ required previous knowledge of another word with $e(q(ID, slave), d(q(ID2, free), d(m, name(q(ID2, free), N))))$ on the right-hand side. Thus our next task was to prove the following.

Lemma 5.10 — No even word with $e(q(ID, slave), d(q(ID2, free), d(m, name(q(ID2, free), N))))$ on the right-hand side is obtainable except by application of the rule

IF $\{X, J\} \subset W$ THEN

$$W := W \cup \{e(m, q(ID, free)), e(m, e(d(m, X), d(q(ID, free), e(d(m, X), d(q(ID, free), d(m, J))))))\}$$

with $X = e(m, q(ID, slave))$ and $J = name(q(ID, free), N)$.

Proof. We ran the narrower on X , where X was assumed to be an even word with $e(q(ID, slave), d(q(ID2, free), d(m, name(q(ID2, free), N))))$ on the right-hand side. Again, we asked the narrower to ignore cases requiring previous knowledge of m or $d(Z, X)$. We came up with the cases (excluding cases involving application of the key-purchase rule with the input listed above) in which prior knowledge of one of the following was required:

- (a) $e(m, e(Q, d(R, e(Q, d(R, d(m, X))))))$
- (b) $e(m, e(Q, d(R, e(Q, d(R, C))))$ where $X = e(m, C)$
- (c) $e(m, e(Q, d(R, e(Q, C))))$ where $X = e(m, e(R, C))$
- (d) $e(m, e(Q, d(R, C)))$ where $X = e(m, e(R, d(Q, C)))$
- (e) $e(m, e(Q, C))$ where $X = e(m, e(R, d(Q, e(R, C))))$
- (f) $e(m, C)$ where $X = e(m, e(R, d(Q, e(R, d(Q, C))))$
- (g) C where $X = e(m, e(R, d(Q, e(R, d(Q, d(m, C))))$

- (h) C where $X = e(Q,C)$
- (i) $e(Q,X)$
- (j) C where $X = d(Q,C)$

or

- (k) $KEYSTATE(B,slave) = d(D,d(m,X))$ and $IDSTATE(B,slave) = [R,D]$
- (l) $KEYSTATE(B,slave) = d(D,C)$ where $X = e(m,C)$ and $IDSTATE(B,slave) = [R,D]$
- (m) $KEYSTATE(B,slave) = C$ where $X = e(m,e(D,C))$ and $IDSTATE(B,slave) = [R,D]$.

Clearly, cases (a) and (b) require prior knowledge of a word with right-hand side $e(q(ID,slave),d(q(ID2,free),d(m,N)))$. So does (c) unless $R = q(ID,slave)$ and $C = d(q(ID2,free),d(m,N))$, in which case $e(m,e(Q,d(R,e(Q,C))))$ must be an odd word. Similar arguments work for cases (d) through (j), except for case (g) where $X = e(m,e(R,d(Q,e(R,d(Q,d(m,name(q(ID2,free),N)))))))$. But in this case (g) is the application of the key-purchase rule that we had ruled out.

Running the narrower program on cases (k) through (m) gives us requirements for prior knowledge of one of the following cases.

For case (k)

- k1. X

For case (l)

- l1. X
- l2. $e(m,e(q(A,B),d(D,e(q(A,B),d(D,C))))))$ where $X = e(m,C)$

For case (m)

- m1. X
- m2. $e(m,e(q(A,B),d(D,e(q(A,B),d(D,d(m,X))))))$

All cases require previous knowledge of an odd word. []

We have shown that the penetrator cannot decrypt a message on a given processor unless that processor has been so authorized by the message's originator, and so Theorem 5.6 is proved.

5.4 Substituting Messages

A message $mess(q(ID,slave),N)$ has been successfully substituted for a message $mess(q(ID2,free),Q)$ if there is a processor $q(B,free)$ with

$IDSTATE(q(B,free)) = [name(q(ID2,free),Q),q(ID2,free)]$
 $RUNSTATE(q(B,free)) = mess(q(ID,slave),N)$.

Accordingly, we ran the narrower on these two state values and found that the following set of conditions must hold immediately prior to attaining these values:

knowledge of $e(X, \text{mess}(q(\text{ID}, \text{slave}), N))$
 $\text{IDSTATE}(q(B, \text{free})) = [\text{name}(q(\text{ID2}, \text{free}), Q), q(\text{ID2}, \text{free})]$
 $\text{KEYSTATE}(q(B, \text{free})) = X$
 $\text{READYSTATE}(q(B, \text{free})) = \text{yes}$
 $\text{RUNSTATE}(q(B, \text{free})) = \text{zero}.$

By running the narrower on the four state values given above, we find that the following must have held previously:

$\text{IDSTATE}(q(B, \text{free})) = [\text{name}(q(\text{ID2}, \text{free}), Q), q(\text{ID2}, \text{free})]$
 $\text{KEYSTATE}(q(B, \text{free})) = d(Y, d(m, \text{name}(q(\text{ID2}, \text{free}), Q)))$
 $\text{READYSTATE}(q(B, \text{free})) = \text{zero}$
 $\text{RUNSTATE}(q(B, \text{free})) = \text{zero}.$

Thus the penetrator can substitute $\text{mess}(q(\text{ID}, \text{slave}), N)$ for $\text{mess}(q(\text{ID2}, \text{free}), Q)$ if and only if he can obtain $e(d(q(\text{ID2}, \text{free}), d(m, \text{name}(q(\text{ID2}, \text{free}), Q))))$, $\text{mess}(q(\text{ID}, \text{slave}), N)$ and cause the state variables of processor $q(B, \text{free})$ to take on the values given above.

By running the narrower on the state values listed above, we find that one of the following is required to hold previously:

- (a) $\text{IDSTATE}(q(B, \text{free})) = [\text{name}(q(\text{ID2}, \text{free}), Q), q(B, \text{free})]$
 $\text{KEYSTATE}(q(B, \text{free})) = \text{zero}$
 $\text{READYSTATE}(q(B, \text{free})) = \text{zero}$
 $\text{RUNSTATE}(q(B, \text{free})) = \text{zero}$
 with knowledge of $\text{name}(q(\text{ID2}, \text{free}), Q)$
- (b) $\text{IDSTATE}(q(B, \text{free})) = [\text{name}(q(B, \text{free}), Q), q(\text{ID2}, \text{free})]$
 $\text{KEYSTATE}(q(B, \text{free})) = \text{zero}$
 $\text{READYSTATE}(q(B, \text{free})) = \text{zero}$
 $\text{RUNSTATE}(q(B, \text{free})) = \text{zero}$
 with knowledge of

$e(m, e(q(B, \text{free}), d(q(\text{ID2}, \text{free}), e(q(B, \text{free}), d(q(\text{ID2}, \text{free}), d(m, \text{name}(q(\text{ID2}, \text{free}), Q)))))))$.

The word $\text{name}(q(\text{ID2}, \text{free}), Q)$ is obtainable when processor $q(\text{ID2}, \text{free})$ publishes $\text{name}(q(\text{ID2}, \text{free}), Q)$, while the word

$e(m, e(q(B, \text{free}), d(q(\text{ID2}, \text{free}), e(q(B, \text{free}), d(q(\text{ID2}, \text{free}), d(m, \text{name}(q(\text{ID2}, \text{free}), Q)))))))$

is obtainable by having processor $q(B, \text{free})$ purchase the rights to $\text{mess}(q(\text{ID2}, \text{free}), Q)$.

When we run the narrower on the state values given in a and b, we are told that they may be obtained if all state values are previously zero and $\text{name}(q(\text{ID2}, \text{free}), Q)$ is previously known, and, in case a, if $e(m, q(B, \text{free}))$ is previously known, and, in case b, if $e(m, q(\text{ID2}, \text{free}))$ is known. Since all $e(m, q(S, T))$ 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 $\text{name}(q(\text{ID2}, \text{free}), Q)$ is obtainable, this set

of state values is obtainable. Thus, working our way back up, we have shown that the set of state values

$$\text{IDSTATE}(q(\text{B}, \text{free})) = [\text{name}(q(\text{ID2}, \text{Q}), q(\text{ID2}, \text{free}))]$$

$$\text{RUNSTATE}(q(\text{B}, \text{free})) = \text{mess}(q(\text{ID}, \text{slave}), \text{N})$$

is obtainable if and only if the word

$$e(d(q(\text{ID2}, \text{free}), d(m, \text{name}(q(\text{ID2}, \text{free}), \text{Q}))), \text{mess}(q(\text{ID}, \text{slave}), \text{N}))$$

is obtainable.

Our next task was to show that

$$e(d(q(\text{ID2}, \text{free}), d(m, \text{name}(q(\text{ID2}, \text{free}), \text{Q}))), \text{mess}(q(\text{ID}, \text{slave}), \text{N}))$$

was unobtainable, and therefore we set out to do this. However, after running the narrower on

$$e(d(q(\text{ID2}, \text{free}), d(m, \text{name}(q(\text{ID2}, \text{free}), \text{Q}))), \text{mess}(q(\text{ID}, \text{slave}), \text{N}))$$

we found that this word *was* obtainable by the penetrator. The scenario is as follows. Processor $q(\text{ID}, \text{slave})$ obtains an encrypted key to $\text{mess}(q(\text{ID2}, \text{free}), \text{Q})$ from processor $q(\text{ID2}, \text{free})$. Processor $q(\text{ID}, \text{slave})$ applies the rule for encrypting messages:

IF $\{A, B, M\} \subset W$ THEN

$$W := W \cup \{e(m, q(\text{ID}, \text{slave})),$$

$$e(m, e(d(m, B), d(q(\text{ID}, \text{slave}), e(d(m, B), d(q(\text{ID}, \text{slave}), d(m, A)))))),$$

$$e(d(q(\text{ID}, \text{slave}), e(d(m, B), e(d(q(\text{ID}, \text{slave}), d(m, A))))), M))\}$$

with input

$$A = e(m, e(q(\text{ID}, \text{slave}), d(q(\text{ID2}, \text{free}), e(q(\text{ID}, \text{slave}), d(q(\text{ID2}, \text{free}), d(m, \text{name}(q(\text{ID2}, \text{free}), \text{Q})))))))$$

$$B = e(m, q(\text{ID2}, \text{free}))$$

$$M = \text{mess}(q(\text{ID}, \text{slave}), \text{N})$$

where A is the encrypted key to $\text{mess}(q(\text{ID}, \text{free}), \text{Q})$. The word output is

$$e(d(q(\text{ID2}, \text{free}), d(m, \text{name}(q(\text{ID2}, \text{free}), \text{Q}))), \text{mess}(q(\text{ID}, \text{slave}), \text{N})).$$

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 $e(m, q(\text{ID}, \text{slave}))$. Moreover, both m and $q(\text{ID}, \text{slave})$ are stored in processor $q(\text{ID}, \text{slave})$. Thus, we can alter the message encryption rule so 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 $\{P, M\} \subset W$ THEN

$$W := W \cup e(d(q(\text{ID}, \text{slave}), d(m, \text{N})), M).$$

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 $\{A, B\} \subset W$ THEN

$$W := W \cup \{e(m, e(d(m, B), d(q(ID, slave), e(d(m, B), d(q(ID, slave), d(m, A))))))\}.$$

These two rules 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 by adding a "dummy word" to the input set of each new rule. Since this word is not used in the formation of new words, this does not affect the rule, and so we do not need to reverify the part of the protocol that was verified by using the original rule.

Now we want to use our revised specification to prove the theorem.

Theorem 5.11 — The word $e(d(q(ID2, free), d(m, name(q(ID2, free), Q))), mess(q(ID, slave), N))$ is unobtainable.

Proof: Now we run the narrower by using the revised specification, on the word

$$e(d(q(ID2, free), d(m, name(q(ID2, free), Q))), mess(q(ID, slave), N)),$$

asking it to ignore cases that require prior knowledge of

$$d(Z, e(d(q(ID2, free), d(m, name(q(ID2, free), Q))), mess(q(ID, slave), N)))$$

or

$$d(q(ID2, free), d(m, name(q(ID2, free), Q))),$$

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

- (a) prior knowledge of
 $e(X, e(d(q(ID2, free), d(m, name(q(ID2, free), Q))), mess(q(ID, slave), N)))$
 and X ;

$e(m, e(B, d(C, e(B, d(C, d(m, e(d(q(ID2, free), d(m, name(q(ID2, free), Q))), mess(q(ID, slave), N))))))))$, or;

- (c) $KEYSTATE(q(E, slave)) = d(X, d(m, e(d(q(ID2, free), d(m, name(q(ID2, free), Q))), mess(q(ID, slave), N)))$ and $IDSTATE(q(E, slave)) = [Y, X]$.

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

$$e(X, d(m, e(d(q(ID2, free), d(m, name(q(ID2, free), Q))), mess(q(ID, slave), N))))$$

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

- (c1) $e(m, e(B, d(C, e(B, d(C, d(m, e(d(q(A, free), d(m, name(q(ID2, free), Q))), mess(q(ID, slave), N))))))))$, or;

- (c2) $e(d(q(ID2, free), d(m, name(q(ID2, free), Q))), mess(q(ID, slave), N))$

is required. This leads us to attempt to verify the unobtainability of all words with

$$e(d(q(ID2, free, d(m, name(q(ID2, free), Q))), mess(q(ID, slave), N)))$$

on the right-hand side.

Lemma 5.12 — No word with $e(d(q(ID2, free, d(m, name(q(ID2, free), Q))), mess(q(ID, slave), N)))$ on the right-hand side is obtainable.

Proof: We ran the narrower on X , where X is assumed to be a word with $e(d(q(ID2, free, d(m, name(q(ID2, free), Q))), mess(q(ID, slave), N)))$ on the right-hand side. Again, we asked the narrower to ignore conditions that required prior knowledge of X or $d(Z, X)$. All of the necessary conditions turned up by the narrower fall into one of the following cases:

Prior knowledge of one of the following:

- (a) $e(m, e(S, d(R, e(S, d(R, d(m, X))))))$
- (b) $e(m, e(S, d(R, e(S, d(R, C))))$ where $X = e(m, C)$
- (c) $e(m, e(S, d(R, e(S, C))))$ and $e(m, R)$ where $X = e(m, e(R, C))$
- (d) $e(m, e(S, d(R, C)))$ where $X = e(m, e(R, d(S, C)))$
- (e) $e(m, e(S, C))$ and $e(m, R)$ where $X = e(m, e(R, d(S, e(R, C))))$
- (f) $e(m, C)$ where $X = e(m, e(R, d(S, e(R, d(S, C))))$
- (g) C where $X = e(m, e(R, d(S, e(T, d(S, d(m, C))))))$
- (h) $e(m, e(q(H, K), D))$ and C where $X = e(D, C)$
- (i) C and $e(m, D)$ where $X = e(d(q(H, slave), D), C)$
- (j) C and D where $X = e(d(q(H, slave), d(m, D)), C)$
- (k) $e(D, X)$ and D
- (l) C and D where $X = e(D, C)$
- (m) C and D where $X = d(D, C)$

In cases (a), (b), (d), (f), (g), (i), (j), (k), and (m), prior knowledge of a word with

$$e(d(q(ID2, free, d(m, name(q(ID2, free), Q))), mess(q(ID, slave), N)))$$

on the right-hand side is clearly required. In cases (c) and (e), it is not if

$$R = d(q(A, free), d(m, name(q(ID2, free), Q))).$$

In case (e), this means prior knowledge of $e(m, e(S, \text{mess}(q(\text{ID}, \text{slave}), N)))$ is required; in case (c), prior knowledge of

$$e(m, d(d(q(\text{ID2}, \text{free}), d(m, \text{name}(q(\text{ID2}, \text{free}), Q))), \\ e(S, d(d(d(q(\text{ID2}, \text{free}), d(m, \text{name}(q(\text{ID2}, \text{free}), Q))), \text{mess}(q(\text{ID}, \text{slave}), N))))))$$

is required. In case (h), prior knowledge of a word with

$$e(d(q(\text{ID2}, \text{free}), d(m, \text{name}(q(\text{ID2}, \text{free}), Q))), \text{mess}(q(\text{ID}, \text{slave}), N))$$

on the right-hand side is not required if

$$D = d(q(\text{ID2}, \text{free}), d(m, \text{name}(q(\text{ID2}, \text{free}), Q))),$$

but in this case prior knowledge of an odd word is required. Case (l) does not require previous knowledge of a word with

$$e(d(q(\text{ID2}, \text{free}), d(m, \text{name}(q(\text{ID2}, \text{free}), Q))), \text{mess}(q(\text{ID}, \text{slave}), N))$$

on the right-hand side if

$$D = d(q(\text{ID2}, \text{free}), d(m, \text{name}(q(\text{ID2}, \text{free}), Q))),$$

but in this case previous knowledge of

$$d(q(\text{ID2}, \text{free}), d(m, \text{name}(q(\text{ID2}, \text{free}), Q)))$$

is required, and this word is unobtainable.

Thus we have proved unobtainability of (c) and (e) when D is an instance of

$$d((q(\text{ID2}, \text{free}), d(m, \text{name}(q(\text{ID2}, \text{free}), Q))).$$

We rechecked the output of the narrower on cases (c) and (e). In all cases of (c) and (d) in which D is an instance of $d((q(\text{ID2}, \text{free}), d(m, \text{name}(q(\text{ID2}, \text{free}), Q)))$, prior knowledge of

$$e(m, D), \text{ that is, } e(m, d(q(\text{ID2}, \text{free}), d(m, \text{name}(q(\text{ID2}, \text{free}), Q))))$$

is also required. But this is an odd word, and we are done. \square

6. COMPARISON WITH OTHER WORK

The approaches taken in the formal verification and analysis of cryptographic protocols fall into four main categories. The first of these is to 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 in Refs. 8, 15, and 16.

Another approach is to develop expert systems that a protocol designer can use to try out various scenarios. This is the approach followed by Millen [17], Longley [18, 19], and to a limited extent by Kemmerer [8]. That this approach can be successfully used to detect previously

undiscovered flaws in a protocol has been shown by Longley and Rigby [18]. 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 cannot be used to provide greater assurance of a protocol's security.

A third approach is to formally model the requirements of a protocol by using logics developed for the analysis of logic and belief. This, for example, is the approach taken in Refs. 20 and 21. This approach is, in a sense, complementary to the approach that we take, and we discuss it in more detail in the next section of this report.

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 [10], upon which our model is based, and the model of Kasami, Yamamura, and Mori [22], also used by Lu and Sundareshan in the verification of a hierarchical key management protocol [23].

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 approach 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.

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 for improvement.

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 also to minimize the effects of system errors and security violations. If the protocol fails to do so, it is considered to be inadequate. For example, the flaw in the Needham-Schroeder protocol [1] discovered by Denning and Sacco [3] required the existence of a previously compromised session key to be exploited.

One of the flaws that our analysis uncovered was of this nature. However, since our analysis was based on a specification of a correctly operating system, there may have been others that we did not catch. One way in which we could catch such flaws would be to specify the incorrect as well as the correct operation of a protocol. For example, we could include rules that describe a processor's state variables taking on incorrect values or rules that describe a penetrator's compromising session keys.

Other questions arose during the modeling of the protocol. One of these was: what actions taken by the penetrator should we consider relevant to the security of a 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 Ref. 5 and included the actions that the penetrator used to exploit its security flaw. However, it would have been 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 so 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 [24]. 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 report 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. To do so, we should turn to other methods. One of these is the methodology presented by Burrows, Abadi, and Needham [20]. In their system, 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 by 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 than our method of analysis provides.

The weakness of the approach of Burrows et al. is that the mapping of the protocol to the assertions and the construction of the inference rules is informal. 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 provided no assistance in determining whether or not the protocol satisfied this assertion. Our analysis showed that the protocol did not.

Thus some sort of layered approach seems to be required using techniques similar to those of Burrows et al. to determine the requirements of a protocol and then using techniques similar to those described in this report to determine whether or not a protocol can meet those requirements. This is in line with current practice in system verification. No one verification technique exists 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 by 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. CONCLUSIONS

In this report 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 by using our techniques. The fact that these techniques exposed two flaws in the protocol provides evidence that these techniques can provide significant assistance in the analysis of protocols.

9. ACKNOWLEDGMENTS

I am grateful to Paul Syverson and Jim Gray for their careful reading of and insightful comments on an earlier version of this report.

REFERENCES

1. R. N. Needham and M. D. Schroeder, "Using Encryption for Authentication in Large Networks of Computers," *Commun. ACM*, 21(12), 993-999 (1978).
2. R. K. Bauer, T. A. Berson, and R. J. Feiertag, "A Key Distribution Protocol Using Event Markers," *ACM Trans. Comp. Sys.* 1, 249-255 (1983).
3. D. E. Denning and G. M. Sacco, "Timestamps in Key Distribution Protocols," *Commun. ACM* 24(8) 533-536 (1981).
4. C. H. Meyer and S. M. Matyas, *Cryptography: A New Dimension in Computer Data Security* (John Wiley and Sons, New York, 1982).
5. G. B. Purdy, G. J. Simmons, and J. A. Studier, "A Software Protection Scheme," Proceedings of the 1988 Symp. on Security and Privacy (IEEE Computer Society Press, Washington, DC, 1982), pp. 99-103.
6. G. J. Simmons, "How to (Selectively) Broadcast a Secret," Proceedings of the IEEE Symposium on Security and Privacy (IEEE Computer Society Press, Washington, DC, 1985), pp. 108-113.
7. J. H. Moore, "Protocol Failures in Cryptosystems," *Proc. IEEE* 76(5) 594-602 (1988).
8. R. A. Kemmerer, "Using Formal Methods to Analyze Encryption Protocols," *IEEE J. Selected Areas in Commun.* 7(4), 448-457 (1989).