

# Using Narrowing in the Analysis of Key Management Protocols

*Catherine Meadows*

Code 5543

Center for Secure Information Technology

Naval Research Laboratory

Washington, DC 20375

## ABSTRACT

In this paper we develop methods for analyzing cryptographic protocols using techniques developed for the solutions of equations in a term rewriting system. In particular, we describe a model of a class of cryptographic protocols and possible attacks on those protocols as term rewriting systems, and we also describe a software tool based on the narrowing algorithm that can be used in the analysis of such protocols. Finally, we use the tool in the analysis of a simple protocol and outline ways in which the tool might be improved to provide greater assistance in the analysis of more complex protocols.

## 1. Introduction

The security of a cryptographic system relies on two things: the security of the cryptographic algorithms used and of the hardware implementing them, and the security of the protocols by which the participants in the system make use of the algorithms to communicate. Since the algorithms and hardware involved are usually developed separately from the protocols, the security of these two parts of the system must to some extent be verified independently. Moreover, it is possible to design protocols with subtle security flaws that are either independent of the crypto-algorithm used (see [1,6,13,18,21] for examples) or result from algebraic properties of the crypto-algorithms involved that can be abstracted away from the definitions of the algorithms themselves (see [18] for a discussion of examples of these). Independent analysis of the protocol provides a means of guaranteeing freedom from such flaws without irrelevant consideration of problems of cryptographic security.

One approach to verifying the security of a protocol independently of the crypto-algorithms used is to represent the protocol as a set of symbolic operations. Necessary properties of the crypto-algorithms (such as the fact that encryption and decryption are inverses), are represented as algebraic properties of the symbols. Analysis of a cryptographic protocol becomes the analysis of properties of a discrete algebraic system. Since the problem of analyzing such a system can become complex, some sort of machine assistance is necessary. Moreover, since in the past much

work has been expended on the development of techniques for machine analysis of similar systems, it seems that a machine system for analyzing protocols from this point of view should be a reachable goal.

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

This approach to the study of cryptographic protocols is not new. Most notably, it has been used by Dolev, Even, Book, and others [2,8-11] to develop algorithms for proving security properties of a class of public-key protocols for which the security problem may be reduced to the problem of determining whether the intersection of two formal languages is nonempty. However, the goal of their research has been to characterize protocols for which efficient algorithms for deciding the security problem may be developed. Our ultimate goal is more similar to the goals of the designers of the software systems presented in

[13,14,16,17], that is, we wish to develop an interactive software tool that can be used to aid the protocol designer in analyzing the security of a wide class of protocols.

We believe that term-rewriting techniques can be a useful aid in the analysis of cryptographic protocols. In support of this, we note that, although none of the software systems listed above explicitly models protocols as term-rewriting systems, cancellation rules are implicit in all these systems, as they are in the Transaction Model, a state machine model of protocols presented by Herzberg and Pinter in [12]. Moreover, in several flawed protocols that have been discussed in the literature, the flaws have arisen as a result of the protocol's properties as a term-rewriting system. For example the flaw of the IBM protocol analyzed in [13] results from the fact that the cryptographic facility involved may be used by a penetrator to generate words in a term-rewriting language that includes the secret keys that that facility is intended to protect. The flaw in the Simmons-Purdy-Studier protocol [22] analyzed by Simmons in [21] arises from a similar use of a set of cryptographic modules. The attacks on the manipulation detection codes discussed in [18] result from a penetrator's ability to use the algebraic properties of exclusive-or to generate illegal words that are nevertheless accepted as legal by the participants in a protocol. The attacks on the notary public scheme implemented using RSA described by Davida [5] and Denning [7] make use of the homomorphic properties of RSA to generate illegally signed documents from legally signed ones; the problem of guaranteeing the safety of protocols using RSA from such attacks is approached from a term-rewriting point of view by Even et. al. in [11].

In the case of other protocols that can not be broken using term-rewriting techniques alone, an attack may nevertheless make use of some of the properties of the protocol as a term-rewriting system. For example, the Needham-Schroeder protocol [19] can be broken by a penetrator who, instead of attempting to find out the secret key passed between participants, induces a set of states in the participants' memory which culminates in a state in which one of the participants accepts as the key an old key supplied by the penetrator [1,6]. However, the penetrator's ability to induce these states depends upon his ability to produce certain words at given points in the protocol, either directly or by inducing the participants to generate the words; thus term-rewriting techniques, although they cannot guarantee the security of such a protocol when used by themselves, can probably be used in conjunction with other techniques to prove security against various classes of attacks.

Given all this, it seems reasonable to argue that it would be useful if the term-rewriting properties of a cryptographic protocol were made explicit and if techniques developed for the analysis of term-rewriting systems were applied to cryptographic protocols.

In this paper we will present a general model for a family of private-key protocols that captures their proper-

ties as term-rewriting systems. We will also present a general strategy for using narrowing, a technique developed for solving equations in term-rewriting languages, to prove security of protocols against a broad spectrum of attacks. Finally, we will show how this strategy can be applied to proving security properties of a simple cryptographic key management system.

## 2. Modeling of Protocols

In this section we develop a general model for private key protocols that use block encryption, prove some properties of the model, and discuss general strategies for proving security of a protocol.

Consider a network in which a number of participants are taking part in a cryptographic protocol. Each participant has the capability of performing a fixed set of operations on a piece of data: for example, he may be able to encrypt or decrypt data, concatenate data into lists, or use a pseudo-random number generator to generate new data from a random seed. Since we wish the protocol to be independent of any cryptographic algorithm chosen, we leave the operations themselves undefined. However, we do require that they have certain properties that can be defined algebraically; for example, in a private key cryptosystem, encryption and decryption with the same key cancel each other out, and in a public key system, encryption and decryption with corresponding public and private keys cancel each other out. Thus it makes sense to represent the results of the operations symbolically as words in an algebraic system, with the cancellation properties interpreted as term rewriting rules. Two words will be considered identical to each other if and only if they are either identical symbolically or they can be reduced to a pair of identical words via the application of a sequence of rewrite rules. For example, if we let  $e(X,Y)$  denote encryption of  $Y$  with key  $X$ , and  $d(X,Y)$  denote decryption of  $Y$  with key  $X$ , and we apply the reduction rules

i.  $d(X,e(X,Y)) \rightarrow Y$ , and;

ii.  $e(X,d(X,Y)) \rightarrow Y$

to the expression  $d(e(d(a,b),d(e(c,d(c,d(a,b),h))))),e(h,k))$ , we have first after applying rule ii to  $e(c,d(c,d(a,b)))$ ,

$$\begin{aligned} & d(e(d(a,b),d(e(c,d(c,d(a,b),h))))),e(h,k)) \rightarrow \\ & d(e(d(a,b),d(d(a,b),h)),e(h,k)), \end{aligned}$$

after applying rule ii to  $e(d(a,b),d(d(a,b),h))$ , we have

$$\begin{aligned} & d(e(d(a,b),d(d(a,b),h)),e(h,k)) \rightarrow \\ & d(h,e(h,k)), \end{aligned}$$

and finally, by a straight forward application of rule i we have

$$\begin{aligned} & d(h,e(h,k)) \rightarrow \\ & k. \end{aligned}$$

Given this framework, a cryptographic protocol can be interpreted as a set of rules for each participant for per-

forming operations on input data. These rules can depend only on the input data itself, or they can also depend on data input at an earlier time. For example, participant A may decide to send data to participant B encrypted under a key received in an earlier step in the protocol.

Consider such a protocol from the point of view of a penetrator who has obtained complete control of the communication medium. Such a penetrator would be able to read all traffic between parties, identify source and destination of all messages, destroy messages, and create his own messages, which could be sent to any destination and be identified as coming from any participant. The penetrator may also be assumed to have access to the encryption and decryption functions, although he may or may not know the secret keys belonging to the participants. Suppose also that the goal of the penetrator is to obtain a secret word or set of words. From the point of view of such a penetrator, the protocol is a set of rules for generating words; in attempting to break the protocol, he is attempting to determine whether or not he can use the rules of the protocol to generate a word that reduces to a secret word. Speaking more mathematically, he is attempting to determine if a certain term-rewriting language contains any words that reduce to a secret word.

In this paper, we will restrict ourselves to private-key protocols, both because less theoretical work seems to have been done in this area, and also because we wish to compare our results with the results obtained by existing software protocol analysis systems, most of which have only been applied to the analysis of private-key systems. In particular, we will apply our results to the IBM system [15] analyzed by Kemmerer in [13] and Longley in [14].

We will assume that the crypto-algorithm used is a block cipher such as DES. We will assume that the penetrator has initial knowledge of a finite set of words, and that all these words are of the same length. Thus, unless type-checking is explicitly required in a protocol, a word can be used interchangeably as a message or as a key for encrypting a message.\* The penetrator has the capability of performing encryption and decryption with a private key, the capability of concatenating words into lists, and the capability of producing words by use of a cryptographically strong pseudo-random number generator. Moreover, he may also have the ability of inducing the other participants in the protocol to perform these operations on inputs he provides them. The operations are represented as follows:

- 1). Encryption of X with key Y is represented as  $e(Y, X)$ .
- 2). Decryption of X with key Y is represented as  $d(Y, X)$ .
- 3). A pseudo-randomly generated number is represented as  $G(i)$ , where  $i$  is an integer and  $G$  is some symbol standing for a particular pseudo-random number generator. In cases where each protocol participant has

one or more such generator, we may represent the number as  $G(A, i)$ , where  $A$  is the ID of a protocol participant.

- 4) If  $N$  is an integer, the successor of  $N$  is represented as  $s(N)$ .

Following the construction given above, the participants in a protocol have the ability of generating a sub-language of a formal language  $L$  formed from an alphabet consisting of a set of letters  $P$  standing for the names of the participant, a set of letters  $M$  containing  $P$  that stand for set of basic words (these may include names, cryptographic master keys, and so on), a set of letters  $G$  standing for pseudo-random number generators, and the operational symbols  $e$ ,  $d$ ,  $s$ , and  $'$ . The productions of  $L$  are given as follows:

$$\begin{aligned} N &\rightarrow 0 \\ N &\rightarrow s(N) \\ T &\rightarrow L \\ T &\rightarrow L.T \\ L &\rightarrow N \\ L &\rightarrow M \\ L &\rightarrow G(N, L) \\ L &\rightarrow e(T, L) \\ L &\rightarrow d(T, L) \end{aligned} \quad (3)$$

We are also given the reduction rules

$$e(X, d(X, Y)) \rightarrow Y \quad \text{and} \quad d(X, e(X, Y)) \rightarrow Y \quad (4)$$

We will also denote by  $L(V)$  the language obtained by substituting the production  $L \rightarrow M \cup V$  for  $L \rightarrow M$  in the definition of  $L$ , where  $V$  is a set of variables.  $L(V)$  obeys the same reduction rules as  $L$ . Thus any theorem we prove about  $L$  will also hold for  $L(V)$ .

The theorem proving strategies we will use for proving properties of protocols require that  $L$  be noetherian and confluent, that is, that every word be reducible to an irreducible word by the application of a finite number of reductions and that, if two different words  $W_1$  and  $W_2$  are obtained from a word  $W$  via reduction, then  $W_1$  and  $W_2$  are both reducible to a common word  $W'$ . That  $L$  is noetherian follows trivially from the fact that each application of a reduction rule to a word reduces its length. Since  $L$  is noetherian, it is enough to prove local confluence in order to prove confluence; that is, it is enough to show that if two different words are obtained from a word using one application each of a reduction rule, then those words are reducible to a common word. We do this in the following lemma:

**Lemma 2.1:** The language  $L$  given in (3) with reduction rules defined in (4) is locally confluent.

*Proof:* Let  $W$  be a word reducible to  $W_1$  and  $W_2$  using one application of a reduction rule in each case. Then  $W$  must contain two subwords,  $g_1 = e(\alpha_1, d(\alpha_1, \beta_1))$  or  $d(\alpha_1, e(\alpha_1, \beta_1))$ , and  $g_2 = e(\alpha_2, d(\alpha_2, \beta_2))$  or  $d(\alpha_2, e(\alpha_2, \beta_2))$ .

\* In the case in which words are not of the same length, we can specify lengths and use concatenation and deconcatenation to build up words of appropriate length for use in encryption, both as keys and text.

If  $g_1$  and  $g_2$  do not overlap, then the result is trivial: reducing  $g_1$  has no effect on  $g_2$  and vice versa, so the word obtained by reducing  $g_1$  and then  $g_2$  is the same as the word obtained by reducing  $g_2$  and then  $g_1$ . Suppose that  $g_1$  and  $g_2$  do overlap. Then one must be a subword of the other. Without loss of generality, we may assume that  $g_2$  is a subword of  $g_1$ . There are two possibilities: either  $g_2$  is a subword of  $\alpha_1$ , or it is a subword of  $\beta_1$ .

If  $g_2$  is a subword of  $\alpha_1$ , let  $\alpha_1'$  be the word obtained by reducing  $g_2$  to  $\beta_2$ . Then  $W_1 = \beta_1$ , and  $W_2 = e(\alpha_1', d(\alpha_1, \beta_1))$  or  $d(\alpha_1', e(\alpha_1, \beta_1))$ . If we reduce  $\alpha_1$  to  $\alpha_1'$  in  $W_2$ , we then have  $W_2$  reducible to  $\beta_1 = W_1$ , and we are done.

If  $g_2$  is a subword of  $\beta_1$ , let  $\beta_1'$  be the word obtained by reducing  $g_2$  to  $\beta_2$ . Then  $W_1 = \beta_1$ , and  $W_2 = e(\alpha_1, d(\alpha_1, \beta_1'))$  or  $d(\alpha_1, e(\alpha_1, \beta_1'))$ . Thus both  $W_1$  and  $W_2$  are reducible to  $\beta_1'$ .

Since  $L(V)$  obeys the same reduction rules as  $L$ , it follows that  $L(V)$  is also noetherian and confluent for any set of variables  $V$ .

We are now ready to give a formal model of a class of cryptographic protocols and to provide a definition of the security problem and discuss strategies for solving it.

**Definition 2.2:** Let  $L$  be a language defined as in (3) with reduction rules as in (4). Let  $W_0$  be a set of words from  $L$ , and let  $S = \langle S_1, \dots, S_n \rangle$  be a tuple of state variables. A *cryptographic protocol*  $(S, R, W_0, L)$  is a set of rules  $R$  each 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  $S := T'$ .

where  $W$  denotes a set of words from  $L$ ,  $T$  is a list of words from  $L(Q)$ , where  $Q$  is some set of variables,  $I$  is a subset of  $L(Q)$ ,  $\text{pred}(I, T)$  is a predicate defined on  $I$  and  $T$ ,  $T' = \langle T'_1, \dots, T'_n \rangle$ , and  $O_i$  and  $T'_j$  are words from  $L(V(I, T, \text{pred}(I, T)))$ , where  $V(E)$  denotes the variables appearing in the list of expressions  $E$ .

In the definition above,  $W_0$  denotes the set of words that the penetrator knows initially,  $I$  represents the message sent by the penetrator to a participant in the protocol (who may be the penetrator himself), and  $\{O_1, \dots, O_k\}$  denotes the message sent by the participant in response to  $I$ .  $W$  denotes the set of words available to the penetrator after the application of a sequence of rules from  $R$ .

In order to aid the exposition, we give a simple example of such a protocol. Consider a protocol consisting of two participants, in which the first participant sends random messages to the second, who encrypts them with a secret key  $ka$ . This can be described using two rules:

Rule 1:

IF  $S_1 = N$  THEN  $W := W \cup \{G(N)\}$  AND  $S_1 := s(N)$ .

Rule 2:

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

The first rule describes the first participant's ability to generate random messages. The second rule describes the fact that the second participant will encrypt any message it receives with  $ka$ .

If we assume that the penetrator knows the encryption algorithm involved, we also need to model his ability to perform encryption and decryption on his own, as follows:

Rule 3:

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

Rule 4:

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

We can also extend the definition of the protocol to include the penetrator's ability to generate random messages, and so on, but for the purposes of our exposition, we will stop here.

We now show how such a definition of a protocol may be used to model the interaction of a penetrator with the participants according to the rules of the protocol.

**Definition 2.3:** Let  $\{V_1, \dots, V_k\}$  be a set of variables and  $\{E_1, \dots, E_k\}$  be a set of expressions. A *substitution*  $\beta = \{V_1/E_1, \dots, V_k/E_k\}$  is a function taking each  $V_i$  to  $E_i$ . If  $E$  is an expression,  $E\beta$  is the expression obtained by substituting  $E_i$  for  $V_i$  in  $E$ .

**Definition 2.4:** Let  $w$  be a word from  $L$  or  $L(V)$ , where  $V$  is some set of variables. We define  $\text{nf}(w)$ , the normal form of  $w$ , to be the word obtained by performing all possible reductions on  $w$ .

Since  $L$  and  $L(V)$  are confluent,  $\text{nf}(w)$  is unique for each  $w$ .

**Definition 2.5:** Let  $(S, R, W_0, L)$  be a cryptographic protocol. Let  $s_0$  be a set of initial values of  $S$ . Let  $((I_1, R_1, s_1, W_1), \dots, (I_m, R_m, s_m, W_m))$  be a sequence such that for each  $j$   $I_j$  and  $W_j$  are sets of irreducible words such that  $I_{j-1}$  is a subset of  $W_j$ ,  $R_j$  is a rule, and  $s_j$  is a set of values for the state variables in  $S$ . We say that such a sequence is a *path through the protocol* if for each  $j$ , there exists a substitution  $\beta_j$  such that  $W_j = W_{j-1} \cup \{\text{nf}(O_1\beta_j), \dots, \text{nf}(O_k\beta_j)\}$  and  $R_j\beta_j =$

IF  $S = s_{j-1}$  AND  $I_j \subset W$  AND  $\text{pred}(I_j, s_{j-1})$  THEN  
 $W := W \cup \{O_1\beta_j, \dots, O_k\beta_j\}$  AND  $S := s_j$ .

In other words  $((I_1, R_1, s_1, W_1), \dots, (I_m, R_m, s_m, W_m))$  is a path through the protocol if  $W_j$  and  $s_j$  are obtained by applying rule  $R_j$  with input words  $I_j$  from  $W_{j-1}$  and input state values  $s_{j-1}$ .

For example, suppose that  $W_0 = \emptyset$  in the protocol defined above and that  $S_1 = 0$  initially. Then the sequence

$\langle \langle \rangle, \text{Rule 1}, \langle 1 \rangle, \{G(0)\} \rangle,$   
 $\langle \langle G(0) \rangle, \text{Rule 2}, \langle 1 \rangle, \{G(0), e(ka, G(0))\} \rangle,$

$(\langle G(0), e(ka, G(0)) \rangle, \text{Rule } 3, \langle 1 \rangle, \{G(0), e(ka, G(0)), e(G(0), e(ka, G(0)))\})$

$(\langle e(G(0), e(ka, G(0))), G(0) \rangle, \text{Rule } 4, \langle 1 \rangle, \{G(0), e(ka, G(0)), e(G(0), e(ka, G(0))), d(e(G(0), e(ka, G(0))), G(0))\})$

is a path through the protocol.

**Definition 2.6:** Let  $(S, R, W_0, L)$  be a cryptographic protocol. Let  $C = C_1 \times C_2$  be a subset of  $L \times L^t$ , where  $t$  is the number of state variables in  $S$ , and  $L^t$  denotes the Cartesian product of  $L$  with itself  $t$  times. We say that the protocol is *secure with respect to C*, or that  $C$  is *unobtainable* if there exists no path  $((I_1, R_1, s_1, W_1), \dots, (I_t, R_t, s_t, W_t))$  such that there exists a member  $(w, s)$  of  $C$  such that  $s_t = s$  and  $w \in W_t$ .

In order to prove a protocol secure against the attacks modeled in the specification, we choose a set  $C$  and attempt to show that  $C$  is unobtainable. Since the definition of  $C$  is in terms of states as well as words obtainable by the penetrator, this definition of security can be used, not only to define security of protocols whose ultimate purpose is to allow the participants to exchange secret information, but authentication protocols whose main purpose is to persuade one or more participants to engage in a certain action as a result of the participating in the protocol. In the case that we are interested only in protecting secret information, we let  $C$  be of the form  $C_1 \times L^t$ .

The basic tool we use for proving security properties of a protocol may be described as follows. Suppose that we wish to determine whether or not the penetrator in our example protocol can derive any irreducible words of the form  $e(Z, ka)$ . We look at the output of each rule in turn, and try to find all substitutions  $\beta$  such that there exists an output word  $O$  such that  $O\beta$  reduces to  $e(Z, ka)\beta$ . Rule 1 gives us no such substitution. Rule 2 gives us  $\beta_1 = (X/ka, Z/ka)$  and  $\beta_2 = (X/d(ka, e(Z, ka)))$ . Rule 3 gives us  $\beta_3 = (Y/d(X, e(Z, ka)))$  and  $\beta_4 = (X/ka, Z/Y)$ . Rule 3 gives us  $\beta_5 = (Y/e(X, e(Z, ka)))$ . We now apply the substitutions to the input words and states of the rules and determine what conditions must hold on the input words and states. Thus, in order to apply Rule 2 with substitution  $\beta_1$ , the penetrator must already know  $ka$ . In order to apply Rule 2 with  $\beta_2$ , the penetrator must already know  $d(ka, e(Z, ka))$ . In order to apply Rule 3, the penetrator must already know  $d(X, e(Z, ka))$  and  $X$  or  $ka$ , depending on the substitution used. Finally, if the penetrator used Rule 5, he must have already known  $e(X, e(Z, ka))$  and  $X$ . These facts allow us to draw general conclusions about what the penetrator needed to know in order to derive  $e(Z, ka)$ , no matter what rule was used: for example, we can conclude he either needed to know  $ka$ ,  $d(W, e(Z, ka))$  or  $e(W, e(Z, ka))$  for some  $W$ . We can then attempt to verify the unobtainability of these words.

In our attempt to determine whether  $e(Z, ka)$  was obtainable, we were applying the following technique. Taking each rule  $R =$

IF  $S = T$  AND  $I \subset W$  AND  $\text{pred}(I, T)$  THEN

$W := W \cup \{O_1, \dots, O_k\}$  AND  $S := T'$

we attempt to describe all substitutions  $\beta$  such that there exists an  $i$  such that  $O_i\beta$  reduces to  $w\beta$ . We can do this if for each  $i$ , we can determine a finite set of substitutions  $\{\beta_{(1,i)}, \dots, \beta_{(s,i)}\}$  such that, if  $\xi$  is a substitution such that  $O_i\xi$  reduces to  $w\xi$ , then there exists a  $j$  such that  $\xi = \beta_{(j,i)}\delta$  for some substitution  $\delta$ . A similar procedure works when we attempt to derive a state.

The process of deriving the substitutions described in the above paragraph, first introduced by Slagle in [23], and known as *narrowing*, is a technique for solving equations in equational systems defined by noetherian confluent term-rewriting languages, such as the language we are using. In the examples presented in this paper, we use a version of the NARROWER algorithm developed by Rety et. al. [20]. We have implemented this algorithm in Prolog [4].

We use our narrowing program to prove properties of protocols in the following manner. We begin by identifying a set  $C_0$  such that the protocol can trivially be shown to be secure with respect to  $C_0$ .  $C_0$  may possibly be the empty set. Once we have identified the set  $C_0$ , we proceed in the following manner. Let  $W_0$  be the set of words available to the penetrator initially, and let  $s_0$  be the initial state. We wish to show that there is no path through the protocol deriving a member of  $C$ . We attempt to accomplish this by induction on the length of the path. If  $C \cap W_0 \times \{s_0\}$  is empty, then the result holds for all paths of length zero. (If the intersection is nonempty, then we have trivially shown that the protocol is insecure.) Suppose that the result holds for all paths of length less than  $m$ . Choose a rule  $R$  of the protocol, and suppose that there is a path of length  $m$  ending in  $(I_m, R, s_m, W_m)$  such that  $C \cap (W_m \times s_m)$  is nonempty, but for all  $i < m$ ,  $C \cap (W_i \times s_i)$  is empty. For each rule  $R$ , we use the narrowing program to determine the conditions on  $I_m$  and  $s_{m-1}$  that must hold in order for this to be true. We let  $C'$  be the subset of  $L \times L^t$  consisting of all elements satisfying the conditions for at least one rule  $R$ . If  $C'$  is a subset of  $C_0 \cup C$ , then we are done, since in the induction hypothesis we assumed that no member of  $C_0 \cup C$  was derivable in less than  $m$  steps. If not, we have several choices: We can attempt to prove the protocol secure with respect to  $C \cup C'$ , we can choose a superset  $C''$  of  $C \cup C'$  and attempt to prove the protocol secure with respect to  $C''$ , or we can break  $C''$  or  $C \cup C'$  down into a set of subsets and attempt to prove the protocol secure with respect to each subset separately.

The decision of how to handle  $C'$  may be tricky. If we break  $C'$  into too many subsets, we run the danger of introducing a combinatorial explosion of paths. However, if we don't break  $C'$  down, it may be too complicated to handle easily. Likewise, if we fail to choose a large enough superset of  $C'$ , we risk the danger of running into an infinite regression. For example, suppose that, in attempt-

ing to verify that a protocol is secure with respect to a word  $x$ , we find that we need to prove it secure with respect to  $e(k,x)$ , and in order to prove it secure with respect to  $e(k,x)$  we find we need to prove it secure with respect to  $e(k,e(k,x))$ , and so on. The obvious solution is to attempt to prove the protocol secure with respect to the superset  $C''$  defined by the language  $R \rightarrow x \mid e(k,R)$ . On the other hand, if we choose  $C''$  to be too large, we run the risk of introducing elements that are derivable in the protocol.

Proving that restrictions exist on the ways in which  $C'$  can be derived may also be useful. For example, we made little progress in our attempt to verify certain security properties of the selective broadcast protocol developed by Simmons in [21] until we realized that, in most of the cases we were looking at, we were attempting to find some word of the form  $d(X,Y)$ , where at most one of the pair  $X,Y$  were known to the penetrator. We then proved as a lemma that it was impossible for the penetrator to derive  $d(X,Y)$  except by use of the rule that describes his ability to perform decryption himself when he knows both  $X$  and  $Y$ . This meant that we had proved that  $d(X,Y)$  was unobtainable unless  $X$  and  $Y$  were already known to the penetrator.

In the next section we will present in detail a simple protocol and a proof of its security against various classes of attacks in order to show how the strategy works in practice.

### 3. Analysis of the IBM Key Management Protocol

In this section we apply our method to the analysis of some of the security properties of a key management protocol developed by IBM and described by Meyer and Matyas in [15]. We chose this example both because its relative simplicity makes it possible to present a detailed outline of its analysis in this paper, and also because two of the software tools mentioned earlier in this paper, the system developed by Kemmerer [13] and that developed by Longley [14], have been used to analyze flaws in a slightly different, insecure version of this protocol.

The IBM protocol was designed to be used in a system consisting of a single host communicating with a set of terminals. The host and each terminal contains a cryptographic facility that is responsible for decryption and encryption. The cryptographic facilities are relatively secure, but have limited storage, while the remainder of the system, although it has more storage available, is considered to be insecure. Thus, although we only allow keys in the clear inside the cryptographic facility, we cannot use it to store keys used in the system. Instead, we encrypt keys using master keys stored in the cryptographic facility. The encrypted keys are stored in the host and in the terminals outside of their cryptographic facilities. This is done as follows. The host's cryptographic facility contains two master keys, which we will call  $mk$  and  $tk$ . The cryptographic facility of terminal  $i$  contains a terminal master key

$t_i$ . Also stored in the host, but outside of the cryptographic facility, are the terminal keys encrypted under master key  $mk$  ( $e(mk,t_i)$ ) and a set of session keys  $s_1$  through  $s_m$ , each encrypted under master key  $tk$ . Session keys are generated dynamically by the host cryptographic facility, using an unspecified pseudo-random number generator. We will also assume that terminal keys are generated the same way. This is not stated explicitly in the description of the protocol; however, it allows us to model the protocol using an arbitrarily large number of terminals.

Since one of the goals of the system is to prevent keys from appearing in the clear outside of the cryptographic facility, it is natural to ask if a penetrator who manages to gain control of the host can, by communicating with the cryptographic facility, obtain a key in the clear. In this section, we will use our methodology to show that, given the assumption that the penetrator communicates only with the host facility, and not any of the terminals, that obtaining a key in the clear outside the facility is impossible. We restrict ourselves to communication with the host facility for the sake of brevity and also because, in the insecure version of this protocol analyzed in [13] and [14], any session or terminal key can be obtained by a penetrator who restricts his activities to communication with the host facility. (In the insecure version of the protocol,  $mk$  and  $tk$  are the same.)

#### 3.1. Definition of the IBM Protocol

There are three state variables, denoted by  $S_{ses}$ ,  $S_{term}$ , and  $S_{pen}$ .  $S_{pen}$  is used to keep track of the last random number generated by the penetrator,  $S_{term}$  is used to keep track of the last terminal key generated by the cryptographic facility, and  $S_{ses}$  is used to keep track of the last session key generated. There are six operations: encryption  $e(X,Y)$ , decryption,  $d(X,Y)$ , the successor operation for integers,  $s(N)$ , and three random number generation operations:  $ses(N)$  (generation of the  $N$ 'th session key),  $term(N)$  (generation of the  $N$ 'th terminal key), and  $pen(N)$  (generation of a random number by the penetrator).

The protocol for communicating with the host cryptographic facility consists of eight rules, defined below:

Rule 1: Generation of New Session Key

IF  $S_{ses} = N$  THEN

$W := W \cup \{e(tk,ses(N))\}$  AND  $S_{ses} := s(N)$ .

Rule 2: Generation of New Terminal Key

IF  $S_{term} = N$  THEN

$W := W \cup \{e(mk,term(N))\}$  AND  $S_{term} := s(N)$ .

Rule 3: Encipher Using Cryptographic Facility

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

Rule 4: Decipher Using Cryptographic Facility

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

Rule 5: Reencipher From Master Key Using Cryptographic Facility

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

$W := W \cup \{e(d(mk, X), d(tk, Y))\}.$

Rules 3 and 4 are used with  $X = e(tk, ses(N))$  and  $Y = \text{message}$  or  $Y = e(ses(N), \text{message})$  to encrypt and decrypt messages to and from the terminals. Rule 5 is used with  $X = e(mk, \text{term}(N))$  and  $Y = e(tk, ses(M))$  in order to generate  $e(\text{term}(M), ses(N))$  which is used to transmit  $ses(N)$  encrypted under  $\text{term}(M)$  to terminal  $M$ .

We also assume that the penetrator has knowledge of the encryption algorithm, and thus is able to perform encryption and decryption on his own.

Rule 6: Encipher

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

Rule 7: Decipher

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

Finally, we assume that the penetrator has the capability of generating random numbers of his own. It turns out that this operation is not used in the analysis, but we include it for the sake of completeness.

Rule 8: Random Number Generation

IF  $S_{\text{pen}} = N$  THEN

$W := W \cup \{\text{pen}(N)\}$  AND  $S_{\text{pen}} := s(N).$

Initially, all state variables are set to 0.  $W_0$  is the empty set.

### 3.2. Analysis of the IBM Protocol

We wish to show that the protocol is secure with respect to the set  $C = \{mk, tk\} \cup \{ses(N) \mid N \geq 0\} \cup \{\text{term}(M) \mid M \geq 0\}$ . We will do this in stages: first we will show that the protocol is secure with respect to  $\{mk, tk\}$ , and then we will show that it is secure with respect to  $\{ses(N) \mid N \geq 0\} \cup \{\text{term}(M) \mid M \geq 0\}$ .

Since  $mk$  and  $tk$  are used only for encryption, it is intuitively obvious that they are unobtainable. We state the unobtainability of  $mk$  and  $tk$  as a general result about cryptographic protocols.

**Definition 3.2.1:** Let  $E$  be an expression in  $L(V_1, \dots, V_k)$ . Let  $a$  be a letter in  $L$ . 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 3.2.2:** Let  $(S, R, W_0, L)$  be a cryptographic protocol. Let  $a$  be a letter in  $L$ . 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.

*Proof:* Clearly, a penetrator cannot obtain a word with  $a$  on the right-hand side unless he inputs a word with  $a$  on the right-hand side. We can thus use induction on the length of the path through the protocol to show that any word with  $a$  on the right-hand side is unobtainable.

We now wish to show that  $ses(N)$  is unobtainable for any  $N$ . When we ran the narrower program on  $ses(N)$ , it produced eight different cases. Upon inspection, however, we found that the eight cases could be divided into three classes. We found that, if a penetrator used a rule to obtain  $ses(N)$ , he must have had prior knowledge of either  $e(X, ses(N))$ ,  $d(X, ses(N))$ , or  $e(tk, d(X, ses(N)))$  for some word  $X$ . Running the narrower program on  $e(tk, d(X, ses(N)))$ , and dividing the output into classes as before, we found that obtaining that word required prior knowledge of  $d(Y, e(tk, d(X, ses(N))))$ ,  $e(Y, e(tk, d(X, ses(N))))$ , or  $e(tk, d(Y, e(tk, d(X, ses(N)))))$  for some word  $Y$ . Running the program on  $d(X, ses(N))$ , we found that obtaining  $d(X, ses(N))$  required prior knowledge of  $ses(N)$ ,  $e(Y, d(X, ses(N)))$ ,  $d(Y, s(X, ses(N)))$ , or  $e(tk, d(Y, d(X, ses(N))))$  for some word  $Y$ .

Those last two facts encouraged us to try to show the unobtainability of the language  $F'$  consisting of the irreducible words\* of the language  $F$  defined by the productions  $F \rightarrow d(L, ses(N)) \mid e(L, F) \mid d(L, F).$

We cannot quite do this, but we can prove the following lemma:

**Lemma 3.2.3:**  $F'$  is unobtainable unless  $ses(N)$  has already been obtained.

*Proof:* We already know that the first production of  $F'$  cannot be obtained unless either  $ses(N)$  or some other word of  $F'$  has been obtained. Running the narrower program on  $e(X, Y)$  and  $d(X, Y)$  where  $Y$  is assumed to be in  $F'$  and checking the output tells us that the same is true of the last two productions.

Lemma 3.2.3 and the output of the narrower program on  $ses(N)$  tell us, that, in order to obtain  $ses(N)$ , the penetrator must know some word of the form  $e(X, ses(N))$ . Running the narrower program on  $e(X, ses(N))$ , we find that, unless  $X = tk$ , obtaining  $e(X, ses(N))$  requires prior knowledge of one of the following:

- 1.a)  $e(Y, e(X, ses(N)))$ ;
- 1.b)  $d(Y, e(X, ses(N)))$ ;
- 1.c)  $e(tk, d(Y, e(X, ses(N))))$ ;
- 1.d)  $e(tk, ses(N))$  and  $e(mk, X)$ , or;
- 1.e)  $e(tk, ses(N))$  and  $Z$  where  $X = d(mk, Z).$

Running the narrower program on  $e(mk, X)$ , we find that, unless  $X = \text{term}(M)$ , obtaining  $e(mk, X)$  requires prior

\* We may restrict ourselves to showing the unobtainability of the irreducible words of  $F$ , since all input words are assumed to be in their irreducible form.

knowledge of  $d(Q, e(mk, X))$ ,  $e(Q, e(mk, X))$ , or  $e(tk, d(Q, e(mk, X)))$  for some word  $Q$ . Running the narrower program on  $d(mk, Z)$ , we find that obtaining  $d(mk, Z)$  requires prior knowledge of  $d(Q, d(mk, Z))$ ,  $e(Q, d(mk, Z))$ , or  $e(tk, d(Q, d(mk, Z)))$  for some word  $Q$ . These facts encourage us to define the following language and verify its unobtainability: Let  $L_1$  denote  $L - \{(term(M) \mid M \geq 0)\}$ , and let  $L_2$  denote  $L_1 - \{tk\}$ . Let  $E$  denote the language defined by  $E \rightarrow e(L_2, ses(N)) \mid e(mk, L_1) \mid d(mk, L) \mid e(L, E) \mid d(L, E)$ .

**Lemma 3.2.4:** Let  $E'$  denote the set of irreducible words of  $E$ . Then  $E'$  is unobtainable unless  $ses(N)$  has already been obtained.

*Proof:* Our previous analysis tells us that the first production of  $E'$  is unobtainable unless either  $ses(N)$  or one of the last four productions has been obtained, and that the next two productions of  $E'$  are unobtainable unless one of the last two productions has been obtained. Finally, analysis of the output of the narrower program for  $e(X, Y)$  and  $d(X, Y)$  where  $Y$  is assumed to be in  $E'$  shows that obtaining such a word requires prior knowledge of a word already in  $E'$ .

Lemma 3.2.4 tells us that, if  $ses(N)$  has not already been obtained, then the only words of the form  $e(X, ses(N))$  that we can obtain are  $e(tk, ses(N))$  and  $e(term(M), ses(N))$ .

When we look back at the output of the narrower program for  $ses(N)$ , we find that, when  $e(X, ses(N))$  is used to find  $ses(N)$ , prior knowledge of either  $X$ ,  $e(tk, X)$  or  $Z$  such that  $X = d(tk, Z)$  is required. This fact and Lemma 3.2.4 tell us that we are reduced to considering the following cases:

- 2.a)  $X = term(M)$  and the penetrator knows  $term(M)$ ;
- 2.b)  $X = tk$  and the penetrator knows  $tk$ ;
- 2.c)  $X = term(M)$  and the penetrator knows  $e(tk, term(M))$ ,  
or;
- 2.d)  $X = tk$  and the penetrator knows  $e(tk, tk)$ .

Cases 2.b and 2.d are already ruled out by Lemma 3.2.2. We are thus left to show that  $term(M)$  and  $e(tk, term(M))$  are unobtainable. The proof that these words are unobtainable is similar to the proof that  $ses(N)$  is unobtainable unless  $term(M)$  or  $e(tk, term(M))$  is obtainable, and we omit it.

#### 4. Directions for Further Research

Our application of the narrower program to the IBM protocol shows that it can be useful in the analysis of cryptographic systems. However, our work also points out many ways in which the program can be improved. For example, in the analysis of the IBM protocol and some other protocols that we have examined, we have again and again encountered the necessity of showing that a set of words belonging to some formal language is unobtainable by the penetrator. So far, we have used the narrower to

generate cases and then inspected them manually to discover patterns suggesting appropriate formal languages and to verify unobtainability of these languages. It seems straightforward to extend the narrower program to provide more assistance in the verification of language unobtainability. It may also be possible to extend the program so that it provides more assistance in the recognition of patterns.

Other improvements are required to deal with more complicated protocols than the IBM protocol examined here. In the IBM protocol, very little is required in terms of state information in order to generate a word. Thus one goal of future research should be to determine how much assistance the narrower program can provide in a protocol in which the participants' actions depend more on the values of state information.

We will also need to be able to represent encrypted and decrypted messages that are more than one block long. Messages are not usually encrypted as separate independent blocks, since doing so would make messages vulnerable to traffic analysis and possibly to attacks based on scrambling and substitutions of message blocks. Thus some dependency between blocks is desirable. For example, consider the cipher block chaining algorithm recommended for use with DES that is described in [15], which makes use of a combination of encryption and exclusive-or to make a block of an encrypted message dependent upon the previous blocks. As it turns out, we do not need to make any extensive changes to the system to make encryption and decryption work properly when cipher block chaining is used. But the fact that the associative, commutative and cancellation properties of exclusive-or have caused problems in some protocols suggests that these properties should also be included in the system. These will not be as straightforward to include, since associative and commutative laws are not reduction rules. However, a substantial body of work exists on the analysis of systems that include commutative and associative laws. Thus we plan to determine how this work can be applied to including the commutative and associative properties of exclusive-or in our system.

Finally, it would be helpful if our model could be extended to provide assistance in the analysis of situations in which a penetrator has previously compromised some information, such as a key distributed in earlier instance of the protocol. It was the consideration of this kind of scenario that led to the discovery of the flaws in the Needham-Schroeder protocol [1,6]. One way of providing such a facility is suggested by Herzberg and Pinter in [12]. In their state machine model of cryptographic protocols, they provide a state variable called the *cost variable*. The cost variable is incremented every time a transaction with a particular cost is performed. Thus a transaction which involves, say, purchasing the right to a cryptographic key may have a certain cost, while a transaction that involves the direct compromise of a key may have a different cost. This allows the notion of compromise of information to be built into a formal model. It should be straightforward



to extend our model to include cost variables.

## 5. Comparison With Other Work

The work described in this paper had its genesis in two areas of research: the work of Dolev, Even, Book and others referred to in the introduction of this paper, and Millen's work on the Interrogator protocol analyzer program [16,17]. From the first we took the idea of modeling a cryptographic protocol as the efforts of a penetrator to generate a word or words in a term-rewriting language, and from the second we took the idea of starting with a state in which a penetrator has broken a protocol and working backwards to determine if we can generate a path of events culminating in that state. However, there are important differences between these and our work. Dolev et. al. are primarily interested in studying a class of protocols for which efficient algorithms for deciding the security problem can be derived. We are more interested in providing automated assistance in the analysis of a wide class of protocols. Millen's Interrogator program is a Prolog "generate and test" program: it generates a large number of paths through the protocol ending in an insecure state; if any of these begin in an initial state, then the protocol is judged to be insecure. Our aim is to prove that, given certain assumptions, a protocol is secure by showing that paths beginning in an initial state and ending in an insecure state cannot be generated.

The model described in this paper depends upon the listing of a finite set of atomic actions performable by a penetrator. In this respect it has certain similarities to other models as well as the ones described above, in particular the Transaction Model of Herzberg and Pinter [12], the Ina Jo specifications of Kemmerer [13], and the expert system devised by Longley [14]. The main difference between the system described in this paper and those listed above is that, instead of presenting the model with a manual security proof of a protocol as do Herzberg and Pinter, or requiring the user to experiment with different attacks as does Longley's system and the executable specifications used by Kemmerer, our system offers automated assistance in proving a protocol secure against a penetrator capable of a finite set of specified actions. Moreover, the techniques offered by the system are chosen from the point of view of compatibility with the particular kinds of things being verified, and thus differ from the more general automatic theorem-proving approach applied in Kemmerer's paper.

The modeling of a cryptographic protocol as a finite set of possible actions by a penetrator is one that is convenient for machine analysis in that it allows one to model a protocol as a finite-state machine. However, such a model is in danger of being incomplete, since it may be possible that there is an action performable by the penetrator, not included in the model of a protocol, that could lead to a compromise. Thus it would be a mistake to assume proving a protocol secure with respect to the model described in this paper will provide a *proof* of the security of a protocol:

the best a protocol designer can do is to prove that his protocol is secure against a penetrator performing a variety of actions that are believed to comprise all those that may be threats to the security of the protocol. On the other hand, this system *does* allow the designer to prove the protocol secure against a penetrator capable of a specified set of actions. Moreover, if a designer discovers new actions that he believes may be used to compromise a protocol, these actions may be added to the specification, either as rewrite rules or as steps of the protocol. Thus this system may be thought of as a kind of "expert system" to be used to prove a protocol secure against a set of attacks that experts have identified as threats.

The approach to modeling used in this paper and the others listed above is not the only one possible for use in the automated analysis of cryptographic protocols. Another promising strategy, taken by Burrows, Abadi, and Needham [3], is to consider the protocol as a set of subprotocols, each of which is mapped to a predicate in some formal belief system. Formal rules of inference are then used to determine whether the set of beliefs guaranteed by the protocol are sufficient to guarantee its security.

This approach avoids the possible incompleteness inherent in our approach of listing the actions available to a penetrator. But the fact that the mapping to the formal predicates is itself informal opens up the possibility of incorrectness. However, it may be possible to apply a system such as the one described in this paper to each subprotocol in order to provide greater, if not absolute, assurance that the mapping is done correctly. Thus the approach described in this paper may be considered to be complementary to the approach taken in [3].

## 6. Acknowledgements

I am grateful to Mark Cornwell and John McLean for their careful comments on an earlier version of this paper, and to Jon Millen for making a copy of his Interrogator program available to me.

## References

1. R. K. Bauer, T. A. Berson, and R. J. Feiertag, "A Key Distribution Protocol Using Event Markers," *ACM Transactions on Computer Systems*, vol. 1, pp. 249-255, August 1983.
2. R. V. Book and F. Otto, "On the Verifiability of Two-Party Algebraic Protocols," *Theoretical Computer Science*, vol. 40, pp. 101-130, 1985.
3. M. Burrows, M. Abadi, and R. Needham, "Authentication: A Practical Study in Belief and Action," in *Proceedings of the 2nd Conference on Theoretical Aspects of Reasoning About Knowledge*, 1988.
4. W. F. Clocksin and C. S. Mellish, *Programming in Prolog*, Springer-Verlag, New York, 1984.
5. G. I. Davida, "Chosen Signature Analysis of the RSA

- (MIT) Public-Key Cryptosystem," Tech Report TR-82-2, Department of Electrical Engineering and Computer Science, University of Wisconsin, Milwaukee, WI, Oct. 1982.
6. D. E. Denning and G. M. Sacco, "Timestamps in Key Distribution Protocols," *Communications of the ACM*, vol. 24 No. 8, pp. 533-536, August 1981.
  7. D. E. Denning, "Digital Signatures with RSA and other Public-Key Cryptosystems," *CACM*, vol. 27, pp. 338-392, April 1984.
  8. D. Dolev, S. Even, and R. Karp, "On the Security of Ping-Pong Protocols," *Information and Control*, vol. 55, pp. 57-68, 1982.
  9. D. Dolev and A. Yao, "On the Security of Public-Key Protocols," *IEEE Transactions in Information Theory*, vol. 29, pp. 198-208, 1983.
  10. S. Even and O. Goldreich, "On the Security of Multi-Party Ping-Pong Protocols," in *Proceedings of the 24th IEEE Symposium on Foundations of Computer Science*, pp. 34-39, IEEE Computer Society Press, Washington, DC, 1983.
  11. S. Even, O. Goldreich, and A. Shamir, "On the Security of Ping-Pong Protocols when Implemented using the RSA," in *Advances in Cryptology - CRYPTO '85*, pp. 58-72, Springer-Verlag, New York, 1986.
  12. A. Herzberg and S. Pinter, "Public Protection of Software," *ACM Transactions on Computer Systems*, vol. 5, No. 4, pp. 371-393, November 1987.
  13. R. A. Kemmerer, "Using Formal Methods to Analyze Encryption Protocols," in *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 134-139, IEEE Computer Society Press, Washington, D. C., 1987.
  14. D. Longley, "Expert Systems Applied to the Analysis of Key Management Schemes," *Computers and Security*, vol. 6, No. 1, pp. 54-67, February 1987.
  15. C. H. Meyer and S. M. Matyas, *Cryptography: A New Dimension in Computer Data Security*, John Wiley & Sons, New York, 1982.
  16. J. K. Millen, "The Interrogator: A Tool for Cryptographic Protocol Security," in *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 134-141, IEEE Computer Society Press, Washington, DC, 1984.
  17. J. K. Millen, S. C. Clark, and S. B. Freedman, "The Interrogator: Protocol Security Analysis," *IEEE Transactions on Software Engineering*, vol. SE-13, No. 2, February 1987.
  18. J. H. Moore, "Protocol Failures in Cryptosystems," *Proceedings of the IEEE*, vol. 76, No. 5, pp. 594-602, May 1988.
  19. R. M. Needham and M. D. Schroeder, "Using Encryption for Authentication in Large Networks of Computers," *Communications of the ACM*, vol. 21 No. 12, pp. 993-999, December 1978.
  20. P. Rety, C. Kirchner, H. Kirchner, and P. Lescanne, "NARROWER: a New Algorithm for Unification and its Application to Logic Programming," in *Rewriting Techniques and Applications, Lecture Notes in Computer Science*, vol. 202, Springer-Verlag, New York, 1985.
  21. G. E. Simmons, "How to (Selectively) Broadcast a Secret," in *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 108-113, IEEE Computer Society Press, Washington, DC, 1985.
  22. G. J. Simmons, G. B. Purdy, and J. A. Studier, "A Software Protection Scheme," in *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 99-103, IEEE Computer Society Press, Washington, DC, 1982.
  23. J. R. Slagle, "Automated Theorem-Proving for Theories with Simplifiers, Commutativity, and Associativity," *JACM*, vol. 21, No. 4, pp. 622-642, October 1974.