ANALYZING ENCRYPTION PROTOCOLS USING FORMAL VERIFICATION TECHNIQUES
(Extended Abstract)

Richard A. Kemmerer

Department of Computer Science
University of California
Santa Barbara, CA 93111

Introduction

Much work has been done in the area of analyzing encryption algorithms, such as DES [Dav 81,Bri 85,BMP 86]. A vast amount of work has also been expended on formally verifying communication protocols [IEE 82,STE 82,RW 83,LS 84,Hol 87]. In contrast, very little work has been devoted to the analysis and formal verification of encryption protocols.

In this paper an approach to analyzing encryption protocols using machine aided formal verification techniques is presented. The idea of the approach is to formally specify the components of the cryptographic facility and the associated cryptographic operations. The components are represented as state constants and variables, and the operations are represented as state transitions. The desirable properties that the protocol is to preserve are expressed as state invariants and the theorems that must be proved to guarantee that the system satisfies the invariants are automatically generated by the verification system.

This approach does not attempt to prove anything about the strength of the encryption algorithms being used. On the contrary, it may assume that the obvious desirable properties, such as that no key will coincidentally decrypt text encrypted using a different key, hold for the encryption scheme being employed.

The following section presents a brief overview of the formal specification language that is used. Next, a sample system is presented along with the desirable cryptographic properties for that system. A formal specification for the example system is then given followed by a discussion about formally verifying and testing encryption protocol specifications. A weakness of the example specification that was discovered through testing the specification is also presented. Finally, a comparison to other work in this area is given, and some thoughts on the usefulness of the approach presented in this paper are discussed.

The Formal Specification Language

Formal specification and verification techniques have become an accepted technique for assuring that a critical system satisfies its requirements. There are a number of formal verification systems available [Rob 79,TE 81,GDS 84,CDL 85,SAM 86]. These systems all use mathematical techniques to guarantee the correctness of the system being designed and implemented. To use these techniques it is necessary to have a formal notation, which is usually an extension of first-order predicate calculus, and a proof theory.

The formal verification system discussed in this paper uses the state machine approach to formal specification. When using the state machine approach a system is viewed as being in various states. One state is differentiated from another by the values of state variables, and the values of these variables can be changed only via well defined state transitions.

The formal specification language that is used in this paper is a variant of Ina Jo*, which is a nonprocedural assertion language that is an extension of first-order predicate calculus. The key elements of the Ina Jo language are types, constants, variables, definitions, initial conditions, criteria, and transforms. A criterion is a conjunction of assertions that specify the critical requirements of a good state. A criterion is often referred to as a state invariant since it must hold for all states including the initial state. An Ina Jo language transform is a state transition function; it specifies what the values of the state variables will be after the state transition relative to their values before the transition. The system being specified can change state only as described by one of the state transforms. A complete description of the Ina Jo language can be found in the Ina Jo Reference Manual [SAM 86].

Before giving a formal specification of the example system a brief discussion of some of the notation is necessary. The following symbols are used for logical operations:

&	Logical AND
$\rightarrow$	Logical implication

In addition there is a conditional form

(if A then B else C)

where A is a predicate and B and C are well-formed terms.
The notation for set operations is:

$\in$	is a member of
$\cup$	set union

* Ina Jo is a trademark of System Development Corporation, a Burroughs Company.

{a,b,...,c} the set consisting of elements a,b,...,and c

{set description} the set described by set description.

The language also contains the following quantifier notation:

$\forall$ for all

$\exists$ there exists

Two other special Ina Jo symbols that may be used are:

N" to indicate the new value of a variable
(eg. N"v1 is new value of variable v1)

T" which defines a subtype of a given type T.

An Example System

The system being used as a pedagogical example in this paper is a single-domain communication system using dynamically generated primary keys and two secret master keys, as described in [MM 80]. The architecture of the system is presented in Figure 1. The terminals communicate directly with the host system, and a new session key (the primary communication key) is dynamically generated by the host for each session. In addition, each terminal has a permanent terminal key that is used by the host to distribute the new session key to a terminal when a new session is initiated. This is the terminal's secondary communication key. Both the terminal keys and the current session keys are stored in encrypted form at the host.

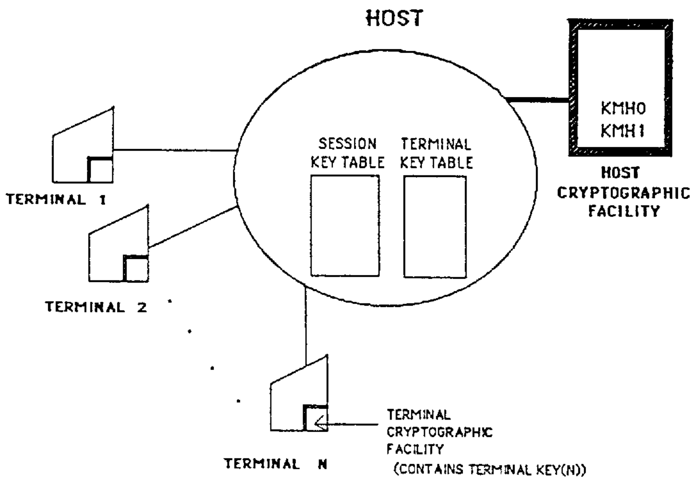
HOST
SESSION KEY TABLE
TERMINAL KEY TABLE
KMH0
KMH1
HOST CRYPTOGRAPHIC FACILITY
TERMINAL 1
TERMINAL 2
TERMINAL N
TERMINAL CRYPTOGRAPHIC FACILITY (CONTAINS TERMINAL KEY(N))

Figure 1 System Architecture

There are two data structures of interest in the host: the terminal key table and the session key table. The ***terminal key table*** is static. Each entry in this table contains the unique terminal key for the corresponding terminal encrypted using a secret master key KMH1. The table looks as follows:

E_{KMH1}(Terminal Key(1))
E_{KMH1}(Terminal Key(2))
:
E_{KMH1}(Terminal Key(i))
:
E_{KMH1}(Terminal Key(n))

In this paper $E_{key-name}$(text) is used to denote text encrypted using the key key-name. Similarly, $D_{key-name}$(text) is used to denote text decrypted using key key-name. Since terminal keys never change, this table is constant for the lifetime of the system.

Unlike the terminal key table, the *session key table* is a dynamic structure. This table is updated each time a new terminal session is started; there is one current session key per terminal. Each entry in the table contains the current session key for the corresponding terminal encrypted using a second secret master key KMH0. The session key table looks as follows:

E_{KMH0}(Session Key(1))
E_{KMH0}(Session Key(2))
:
E_{KMH0}(Session Key(i))
:
E_{KMH0}(Session Key(n))

No terminal key, session key, nor either master key is in the clear in the host. To store the two masters keys a *cryptographic facility* is connected to the host. This facility may be accessed only through the limited cryptographic operations that are provided. In addition, the facility is assumed to be housed in a tamper-sensing container, such as the one described by Simmons [Sim 85], so that the vital information it contains is physically protected. The operations provided by the cryptographic facility are encipher data (ECPH), decipher data (DCPH), and reencipher from master key (RFMK). The interaction between the host and its cryptographic facility is shown in Figure 2.

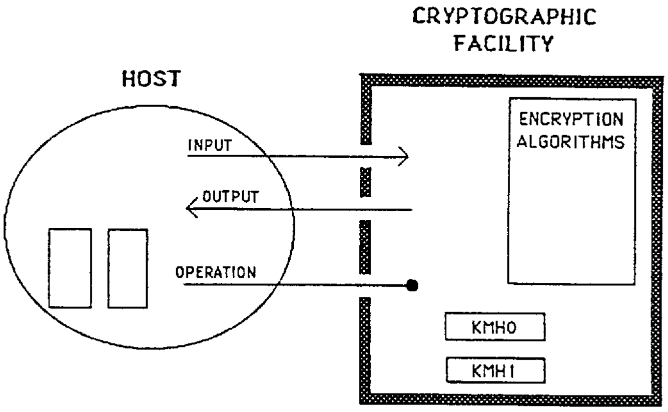

Figure 2 Host and Cryptographic Facility

The *encipher* operation is used when the host wants to send an encrypted message to a terminal. The host provides the clear text message (msg) along with the encrypted form of the appropriate terminal's current session key, E_{KMH0}(Session Key(i)), to the cryptographic facility and is returned the message encrypted using the terminal's session key. Figure 3 illustrates this process.

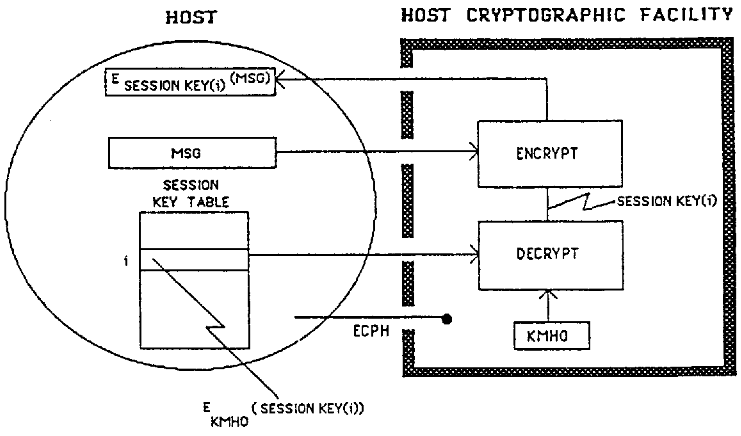

Figure 3 The Encipher Operation (ECPH)

When the host receives an encrypted message from terminal i it uses the *decipher* operation provided by the cryptographic facility in a similar manner to get the message in the clear. That is, the decipher operation must first decrypt the key presented and then use the result to decipher the text presented. Figure 4 illustrates the decipher process.

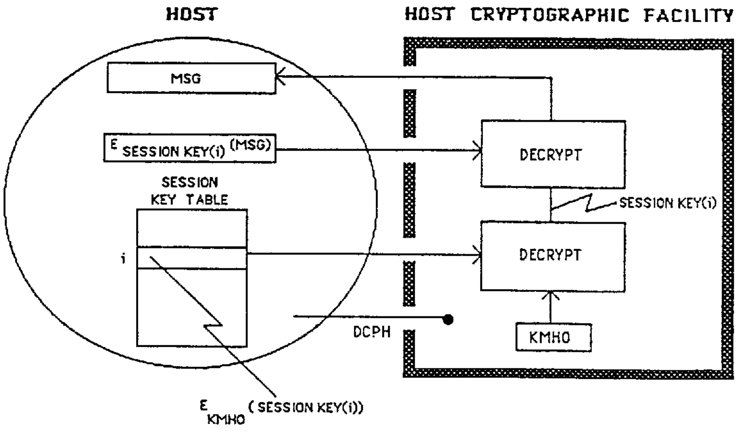

Figure 4 The Decipher Operation (DCPH)

Each time a terminal initiates a session with the host a new session key is needed. Therefore, the host needs access to a *generate session key* operation. It is not necessary, however, to have session key generation be an operation provided by the cryptographic facility. The seemingly contrary requirements of having the host generate the session key and having no key in the clear outside the cryptographic facility are resolved by having the host generate an encrypted version of the session key. The entry in the host's session key table that corresponds to the terminal requesting a session is then replaced by the encrypted key. Meyer and Matyas describe a method of accomplishing this by using a pseudo-random number generator to generate a value that is interpreted as the new session key encrypted using the secret master key KMHO [MM 80].

Since the requesting terminal is also sent a copy of the session key, it is necessary to have an operation to translate the new session key enciphered using the KMHO master key to a form enciphered using the requesting terminal's terminal key. The *reencipher from master key* operation provides this service. It takes two keys as input, decrypts one using KMH1, decrypts the other using KMHO, and then uses the result of the first decryption to encipher the result of the second decryption. Figure 5 illustrates this process.

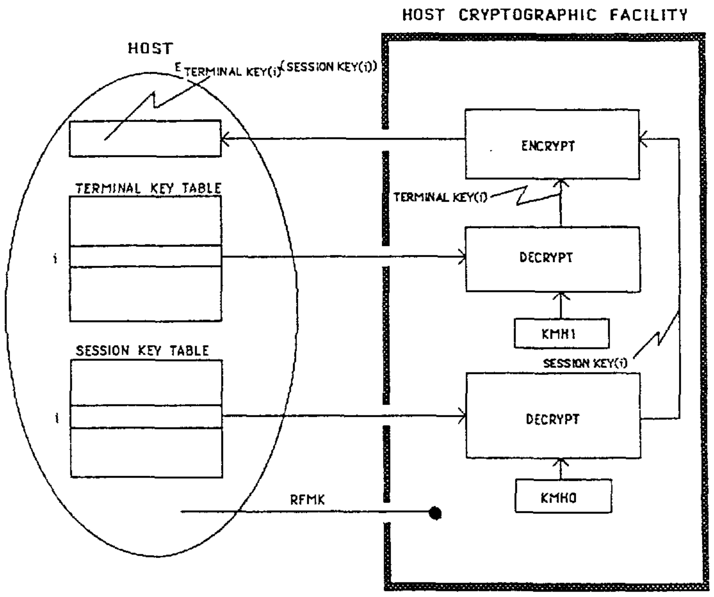

Figure 5 Reencipher From Master Key (RFMK)

In addition to the host's cryptographic facility, each terminal is assumed to have a cryptographic facility that contains its permanent terminal key and that provides operations for encrypting and decrypting messages.

Operations for setting the secret master keys in the host's cryptographic facility or the secret terminal keys in the terminal cryptographic facilities have intentionally been avoided in this paper. It is assumed that these secret keys are distributed by courier or some other trusted means.

An assumption of this system is that the intruder can obtain any information communicated between the host and the terminals. In addition, the intruder can masquerade as an authorized user, and he/she can invoke any of the operations of the host's cryptographic facility.

An obvious desirable property that one may wish to verify about this system is that no clear key exists outside the cryptographic facilities of the host and terminals.

Formal Specification of the Example System

The complete Ina Jo specification for the example system is presented in the appendix. In this section the important aspects of the specification are discussed.

In the example system each terminal has a constant terminal key. This is represented in the model by the Ina Jo constant

Terminal_Key(Terminal_Num): Key.

Similarly, each terminal's session key, which is dynamic, is represented by the Ina Jo variable

Session_Key(Terminal_Num): Key.

As the terms infer, an Ina Jo *constant* is unchanged from state to state and an Ina Jo *variable* may change from state to state. It is the value of the state variables that differentiate one state from another.

The other state variables in the specification are Keys_Used and Intruder_Info, which are both of type information. Keys_Used keeps track of all of the keys that have ever been used by the system. The Intruder_Info variable keeps track of all of the information that the intruder has access to. This includes the contents of the encrypted key tables as well as any information communicated between the host and the terminals.

The four cryptographic operations for the example system are represented by the Ina Jo *transforms* ECPH, DCPH, RFMK, and Generate_Session_Key. The first three correspond to the operations provided by the host's cryptographic facility and the last is provided by the host itself. Since only traffic that is a key or an encrypted form of a key is of interest, there is no need to model any of the send or receive text operations. Also, because it is assumed that the intruder cannot correctly guess random text that corresponds to some encrypted key, the ECPH, DCPH, and RFMK operations change state only when they are invoked with information that the intruder has available. Therefore, each of these operations is written using a conditional form where there is no state change if the information provided is not available to the intruder (i.e., not part of Intruder_Info).

The constants Encrypt and Decrypt represent the encryption and decryption algorithms, respectively. They both take a key and text pair and return text. Properties of the encryption and decryption algorithms are represented in the specification as axioms. An Ina Jo *axiom* is an expression of a property that is assumed. Thus, these qualities are assumed about the algorithms. For instance, one could express the fact that the encryption and decryption functions were commutative by using the axiom

AXIOM $\forall$ t:Text, k1,k2:Key (Encrypt(k1,Decrypt(k2,t))=Decrypt(k2,Encrypt(k1,t)))

Similarly, the following axioms express the properties that no key is an identity function for any text and that no key will correctly decrypt text encrypted using another key.

AXIOM $\forall$ t:Text, k1:Key (Encrypt(k1,t) $\neq$ t)

and

AXIOM $\forall$ t:Text, k1,k2:Key (k1 $\neq$ k2 $\rightarrow$ Decrypt(k2,Encrypt(k1,t)) $\neq$ t)

Note that all of these assumptions are not expressed in the example specification.

The fact that the intruder receives all of the information that is communicated between the host and the terminals is expressed in the Generate_Session_Key transform where the intruder's information is enhanced with the new session key encrypted using the terminal key of the requesting terminal. Also, since the intruder can masquerade as an authorized user, whenever one of the cryptographic operations is invoked the intruder's information is updated with the new information that is produced.

The critical requirements that the system is to satisfy in all states are expressed in the *criterion* clause of the formal specification. For the example system the criterion states that any key that the intruder has (i.e., any key contained in the set Intruder_Info) must not be a key that was used by the system (i.e., a key in the set Keys_Used). Note that this includes keys used in the past as well as those presently being used. The criterion is expressed as follows:

CRITERION $\forall$ k:Key (k $\in$ Intruder_Info $\rightarrow$ k $\notin$ Keys_Used)

The *initial* clause describes the requirements that must be satisfied when the system is initialized. For the example system the initial value of the Keys_Used variable is the set of keys currently being used (i.e., all terminal keys and session keys as well as both master keys). The initial value of the Intruder_Info variable is the appropriate encrypted versions of the terminal and session keys. Because the keys are not required to have any particular value, their values are not specified in the initial clause. However, since the desirable property is for the intruder to never have any keys in the clear, the last conjunct of the initial clause states that none of the encrypted values of the keys can be coincidentally equal to a key being used. That is,

$\forall$ k1,k2:Key (k1 $\in$ Intruder_Info & k2 $\in$ Keys_Used $\rightarrow$ k1 $\neq$ k2).

The need for this additional requirement is discussed in the next section.

To verify that the system specified satisfies the invariant requirements, as expressed in the criteria, two types of theorems are generated. The first states that the initial state satisfies the invariant and the second, which is generated for each transform, states that if the state where the transform is fired satisfies the invariant, then the resultant state will also satisfy the invariant. Thus, by induction all reachable states will satisfy the invariant.

If one can verify the theorems generated, then any system that is consistent with the specification will preserve the invariant. The reader should note that for a system to be consistent with the specification its encryption algorithms must satisfy the axioms stated about encrypt and decrypt.

Formally Verifying the Specification

After the formal specification is completed one can verify the theorems that are generated to check if the critical requirements (Ina Jo criteria) are satisfied. If the theorems are verified and the encryption algorithms satisfy the assumed axioms, then the system will satisfy its critical requirements.

Because the axioms represent the properties that the encryption algorithms are to satisfy, one can verify the system assuming the use of a different encryption scheme by replacing the current axioms with axioms that express the properties of the new encryption scheme.

An advantage of expressing the system using formal notation and attempting to prove properties about the specification is that if the generated theorems cannot be proved the failed proofs often point to weaknesses in the system or to an incompleteness in the specification. That is, they often indicate the additional assumptions required about the encryption algorithm (i.e., missing axioms), weaknesses in the protocols, or missing constraints in the specification. For example, the original specification for the example system did not include the third conjunct that is now in the initial clause. However, without this conjunct the initial clause was not strong enough to imply the invariant. After analyzing the failed proof for some time the possibility of an encrypted version of a key being coincidentally identical to another key was apparent. By adding the third conjunct to the initial clause the problem was avoided. This was a reasonable change to make to the specification since the the occurrence of coincidental values is easy to check when the system is initialized.

Being aware of the coincidental key value problem in the initial clause resulted in a strengthening of the specification for the generate session key operation. That is, the requirements

    Encrypt(KMH0,k) ∉ Keys_Used

and

    Encrypt(Terminal_Key(Ter),k) ∉ Keys_Used

were added to the formal specification to prevent the encrypted value chosen as a new session key from being coincidentally equal to the value of a key that had been used in the past. This requirement is likely to be harder to realize in an actual system since it requires recording information about all keys that have ever been used.

Testing the Formal Specification

There is a specification execution tool for the Ina Jo language called Inatest [EK 85]. This tool allows Ina Jo specifications to be analyzed by symbolically executing the formal specifications. With the Inatest tool it is possible to interactively introduce assumptions about the system, execute sequences of transforms, and check the results of these executions. This provides the user with a rapid prototype for testing properties of the cryptographic facilities [Kem 85].

Using the Inatest tool revealed the following weakness in the example formal specification. If the secret master keys, KMH0 and KMH1, are equal, then the intruder can obtain a session key in the clear. This flaw is demonstrated by first invoking the Generate_Session_Key transform, which generates a new session key, k. This key is communicated to the requesting terminal (encrypted using the terminal key of the requesting terminal) at the start of the current session; therefore, the encrypted key becomes part of the intruder's information. The DCPH transform is then invoked using the encrypted terminal key from the host's terminal key table and the intercepted session key encrypted using the the requesting terminal's terminal key. Figure 6 illustrates the result of executing the DCPH transform on the two encrypted keys.

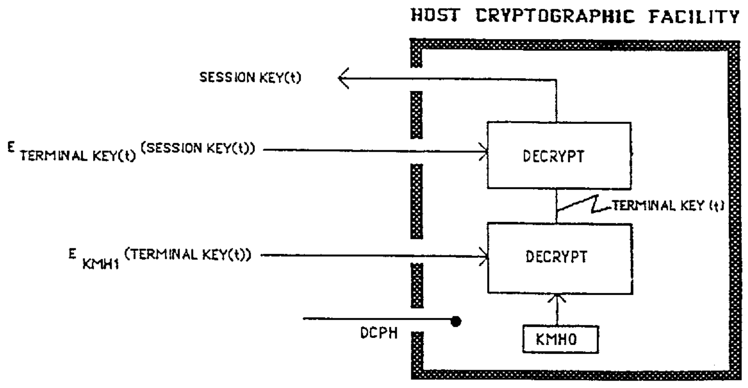

Figure 6 Protocol Flaw Using DCPH

When using the Inatest tool to test a formal Ina Jo specification the user defines a start state, a sequence of transforms (with the appropriate actual parameters) to be executed, and a desired resultant state. To test this weakness the default start state, which is the initial state, was used

Keys_Used=Terminal_Keys ∪ Session_Keys ∪ {KMH0,KMH1}

& Intruder_Info =

{ks:Key (∃ t:Terminal_Num (ks=Encrypt(KMH0,Session_Key(t))))}

∪ {kt:Key (∃ t:Terminal_Num (kt=Encrypt(KMH1,Terminal_Key(t))))}

& ∀ k1,k2:Key (k1 ∈ Intruder_Info & k2 ∈ Keys_Used → k1≠k2)

The sequence of transforms to be executed consists of the Generate_Session_Key transform followed by the DCPH transform. The parameters of the DCPH transform are the encrypted terminal key for terminal t and the current session key for terminal t encrypted using the terminal key for terminal t. Both keys are known to be part of the intruder's information. The first is from the terminal key table, and the second was sent to terminal t when the current session was started. Letting k represent the key that results from executing the Generate_Session_Key transform on behalf of terminal t, the sequence is,

```
        Generate_Session_Key(t)
followed by
        DCPH(Encrypt(KMH1,Terminal_Key(t)), Encrypt(Terminal_Key(t),k)),
```

The desired resultant state requires that the key for terminal t that was generated by the Generate_Session_Key transform be part of the intruder's information. This requirement is expressed as:

```
      k ∈ Intruder_Info
```

This is a clear violation of the security requirement since one of the assumptions included in the start state is that k is one of the keys used by the system.

By expanding Generate_Session_Key one gets

```
 ∃ k:Key ∀ t:Terminal_Num (
    Encrypt(KMH0,k) ∉ Keys_Used
  & Encrypt(Terminal_Key(Ter),k) ∉ Keys_Used
  & k ∉ Keys_Used
  & N"Session_Key(t) =
      if t=Ter
        then k
        else Session_Key(t)
  & N"Keys_Used = Keys_Used ∪ {k}
  & N"Intruder_Info = Intruder_Info ∪
        {Encrypt(KMH0,k),Encrypt(Terminal_Key(Ter),k))}
```

The existential is instantiated to k and the resulting information is combined with the start state information.

Next, by expanding the DCPH transform one gets

```
 N"Intruder_Info =
  if  Encrypt(Terminal_Key(t),k) ∈ Intruder_Info
   & Encrypt(KMH1,Terminal_Key(t)) ∈ Intruder_Info
     then Intruder_Info ∪
        {Decrypt(Decrypt(KMH0,Encrypt(KMH1, Terminal_Key(t))),Encrypt(Terminal_Key(t),k))}
     else Intruder_Info.
```

Since both keys are part of the intruder's information this can be reduced to

```
 N"Intruder_Info = Intruder_Info ∪
```

{Decrypt(Decrypt(KMH0,Encrypt(KMH1, Terminal_Key(t))),Encrypt(Terminal_Key(t),k))}.

Since the start state specifies that KMH0=KMH1, KMH1 can be substituted for KMH0 in the innermost Decrypt yielding

N"Intruder_Info = Intruder_Info ∪
{Decrypt(Decrypt(KMH1,Encrypt(KMH1, Terminal_Key(t))),Encrypt(Terminal_Key(t),k))}.

Then by applying the first axiom the expression reduces to

N"Intruder_Info = Intruder_Info ∪
{Decrypt(Terminal_Key(t),Encrypt(Terminal_Key(t),k))}.

Applying the first axiom again yields

N"Intruder_Info = Intruder_Info ∪ {k}.

The desired result follows directly.

This is a well known weakness of using a single master key that is presented in [MM 80]. To strengthen the specification to avoid this particular problem one needs to add the axiom

AXIOM KMH0 $\neq$ KMH1.

Comparison to Previous Work

As was mentioned in the introduction very little work has been devoted to the analysis and formal verification of encryption protocols. In particular, formal verification techniques have not been used in the analysis efforts that have been reported. A notable exception is the Interrogator work of Millen [MCF 87].

The work reported in this paper differs from Millen's work in that the goal of the work being reported is to use existing formal verification tools to formally verify that an encryption protocol specification satisfies its security requirements (as expressed in the Ina Jo criteria). This is accomplished by using the existing Formal Development Methodology (FDM) tool suite and treating the encryption protocol specification like any Ina Jo formal specification.

The two efforts are similar in that they both use a formal notation to express the protocol (The Interrogator uses a Prolog specification.). The use of the Inatest tool for testing particular scenarios is also similar to the use of the Interrogator tool. However, there are at the same time major differences between using Inatest to test a protocol and using the Interrogator. When using Millen's Interrogator the prolog program exhaustively searches for penetrations. Inatest, in contrast, does not search through a large number of scenarios to detect a vulnerability. It is the task of the human analyzer to come up with a possible scenario that is then checked using the Inatest tool to execute the formal specification. Inatest does not direct the analyst to determine what tests to try; it merely aids the analyst by keeping track of state information and performing reductions when possible. Finally, the Interrogator tool was built explicitly for analyzing encryption protocols, but Inatest was built to execute any Ina Jo specification. As a result, the Interrogator includes a more sophisticated display that dynamically illustrates the progress of the protocol being tested.

Conclusions

This paper has proposed an approach to analyzing encryption protocols using existing formal specification and verification techniques. The approach assumes the availability of encryption algorithms that satisfy the properties expressed in the axioms.

An example system was specified using the Ina Jo specification language. Some problems discovered when attempting to prove the original specification are discussed, and a weakness in the formal specification that was revealed by using an interactive testing tool was presented.

One of the advantages of this approach is that the cryptographic facility can be analyzed assuming different encryption algorithms by replacing the set of axioms that express the properties assumed about the encryption algorithms with a new set of axioms that express the properties of a different encryption algorithm.

Another advantage is that the properties of a cryptographic facility can be tested before it is built by using the formal specification and the available interactive testing tool as a rapid prototype.

The flaw that was confirmed by using the Inatest tool was a previously known weakness of the protocol being analyzed. The true worth of the proposed approach will be established only when a flaw can be discovered in a protocol that has been previously assumed to be secure.

References

[Bri 85] Brickell, Ernest F. "Breaking Iterated Knapsacks," *Advances in Cryptology: Proceedings of Crypto 84*, Lecture Notes in Computer Science, Springer-Verlag, New York, 1985.

[BMP 86] Brickell, E.F., J.H. Moore, and M.R. Purtill, "Structure of the S-Boxes of the DES," *Proceedings of CRYPTO 86*, Santa Barbara, California, August 1986.

[CDL 85] Crow, J., D. Denning, P. Ladkin, M. Melliar-Smith, J. Rushby, R. Schwartz, R. Shostak, and F. von Henke, "SRI Verification System Version 2.0 Specification Language Description," SRI International Computer Science Laboratory, Menlo Park, California, November 1985.

[Dav 81] Davies, Donald W., "Some Regular properties of the 'Data Encryption Standard' Algorithm," Proceedings of CRYPTO 81, *Advances in Cryptography*, Department of Electrical and Computer Engineering Report, ECE 82-04, Santa Barbara, California, August 1986.

[EK 85] Eckmann, Steven T., and Richard A. Kemmerer, "INATEST: An Interactive Environment for Testing Formal Specifications," Third Workshop on Formal Verification, Pajaro Dunes, California, February, 1985, *ACM - Software Engineering Notes*, Vol. 10, No. 4, August 1985.

[GDS 84] Good, D.I., B.L. DiVito, and M.K. Smith, "Using the Gypsy Methodology," Institute For Computing Science, University Of Texas, June 1984.

[Hol 87] Holzmann, Gerard J., "Automated Protocol Validation in Argos: Assertion Proving and Scatter Searching," *IEEE Transactions on Software Engineering*, Vol. SE-13, No. 6, June 1987.

[IEE 82] Sunshine, Carl A. (Editor), Special Issue on Protocol Specification and Verification, *IEEE Transactions on Communications*, Vol. COM-30, No. 12, December 1982.

[Kem 85] Kemmerer, Richard A., "Testing Formal Specifications to Detect Design Errors," *IEEE Transactions on Software Engineering*, Vol. SE-11, No. 1, January 1985.

[LS 84] Lam, Simon S., and A. Udaya Shankar, "Protocol Verification Via Projections," *IEEE Transactions on Software Engineering*, Vol. SE-10, No. 4, July 1984.

[MM 80] Meyer, Carl H., and Stephen M. Matyas, *Cryptography*, John Wiley, 1980.

[MCF 87] Millen, Jonathan K., Sidney C. Clark, and Sheryl B. Freedman, "The Interrogator: Protocol Security Analysis," *IEEE Transactions on Software Engineering*, Vol. SE-13, No. 2, February 1987.

[Rob 79] Robinson, L., "The HDM Handbook, Vol I: The Foundations Of HDM," Computer Science Laboratory, SRI International, Menlo Park, California, June 1979.

[RW 83] Rudin, H., and C.H. West (Editors), *Protocol Specification, Testing, and Verification III* Elsevier Science Publishers B.V., North-Holland, 1983.

[SAM 86] Scheid, J., S. Anderson, R. Martin, and S. Holtsberg, "The Ina Jo Specification Language Reference Manual," SDC document, System Development Corporation, Santa Monica, California, January 1986.

[Sim 85] Simmons, G.J., "How to (Selectively) Broadcast a Secret," Proceedings IEEE Symposium on Security and Privacy, Oakland, California, April 1985.

[STE 82] Sunshine, Carl A., David H. Thompson, Roddy W. Erickson, and Susan L. Gerhart, "Specification and Verification of Communication Protocols in AFFIRM Using State Transition Models," *IEEE Transactions on Software Engineering*, Vol. SE-8, No. 5, September 1982.

[TE 81] Thompson, D.H. and R.W. Erickson, eds. "Affirm Reference Manual," USC Information Sciences Institute, Marina del Rey, California, February 1981.

Appendix Formal Specification of the Example System

```
SPECIFICATION Crypto
LEVEL Top_Level

TYPE
   Text,
   Key subtype Text,
   Pos_Integer = T" i:Integer (i>0),
   Information = Set Of Text

CONSTANT
   Num_Terminals: Pos_Integer,
   KMH0, KMH1: Key,
   Encrypt(Key,Text): Text,
   Decrypt(Key,Text): Text

TYPE
   Terminal_Num = T" t:Pos_Integer (t<=Num_Terminals)

CONSTANT
   Terminal_Key(Terminal_Num): Key,
   Terminal_Keys: Information =
      {k:Key ( ∃ t:Terminal_Num (k=Terminal_Key(t)))}

AXIOM
   ∀ t:Text, k1,k2:Key (
      k1=k2 → Decrypt(k1,Encrypt(k2,t))=t )

AXIOM
   ∀ t:Text, k1,k2:Key (
      k1=k2 → Encrypt(k1,Decrypt(k2,t))=t )

VARIABLE
   Session_Key(Terminal_Num): Key,
   Keys_Used: Information,
   Intruder_Info: Information

DEFINE
   Session_Keys: Information ==
      {k:Key ( ∃ t:Terminal_Num (k=Session_Key(t)))}

CRITERION
   ∀ k:Key (k ∈ Intruder_Info → k ∉ Keys_Used)

INITIAL
   Keys_Used=Terminal_Keys ∪ Session_Keys ∪ {KMH0,KMH1}
 & Intruder_Info =
     {ks:Key ( ∃ t:Terminal_Num (ks=Encrypt(KMH0,Session_Key(t))))}
   ∪ {kt:Key ( ∃ t:Terminal_Num (kt=Encrypt(KMH1,Terminal_Key(t))))}
 & ∀ k1,k2:Key (k1 ∈ Intruder_Info & k2 ∈ Keys_Used → k1≠k2)
```

```
Transform ECPH(K:Key, T:Text)  EXTERNAL
 Effect
   N"Intruder_Info =
       if T ∈ Intruder_Info & K ∈ Intruder_Info
          then Intruder_Info ∪ {Encrypt(Decrypt(KMH0,K),T)}
          else Intruder_Info

Transform DCPH(K1:Key, T1:Text)  EXTERNAL
 Effect
   N"Intruder_Info =
      if T1 ∈ Intruder_Info & K1 ∈ Intruder_Info
          then Intruder_Info ∪ {Decrypt(Decrypt(KMH0,K1),T1)}
          else Intruder_Info

Transform RFMK(K1:Key, K2:Key)  EXTERNAL
 Effect
   N"Intruder_Info =
     if K1 ∈ Intruder_Info & K2 ∈ Intruder_Info
        then Intruder_Info ∪ {Encrypt(Decrypt(KMH1,K1), Decrypt(KMH0,K2))}
        else Intruder_Info

Transform Generate_Session_Key(Ter:Terminal_Num)  EXTERNAL
 Effect
    ∃ k:Key ∀ t:Terminal_Num (
        Encrypt(KMH0,k) ∉ Keys_Used
      & Encrypt(Terminal_Key(Ter),k) ∉ Keys_Used
      & k ∉ Keys_Used
      & N"Session_Key(t) =
          if t=Ter
             then k
             else Session_Key(t)
      & N"Keys_Used = Keys_Used ∪ {k}
      & N"Intruder_Info = Intruder_Info ∪
             {Encrypt(KMH0,k),Encrypt(Terminal_Key(Ter),k)})

END Top_Level
END Crypto
```