Formal Characterization and Automated Analysis of Known-Pair and Chosen-Text Attacks

Stuart G. Stubblebine   Catherine A. Meadows

Abstract— **Formal methods have been successfully applied to exceedingly abstract system specifications to verify high level security properties such as authentication, key exchange, and fail-safe revocation. Furthermore, considerable research exists on evaluating particular ciphers and secure hash functions used to implement high level security properties. However, verifying that less abstract system specifications satisfy low level security properties has been largely impractical. This is evidenced by innumerable system vulnerabilities where high level properties are not attained due to failed assumptions of low level properties.**

This paper presents ongoing work on investigating known pairs and chosen text using the NRL Protocol Analyzer. We give a formal characterization of known and chosen pairs, and translate it to necessary and sufficiency conditions in the NRL Protocol Analyzer model. It is the first work the authors are aware of automatically discovering known-pair and chosen-text attacks. We describe the use of the Analyzer to rediscover attacks, to find new variants of attacks on an early version of the ESP protocol, and to show how our experience in using it has led us to refine our model. This was the first use of the Analyzer to model protocols at such a low level of abstraction.

Keywords— **formal verification methods, security assessment, security and privacy, chosen plaintext, known plaintext, network security, cryptography, cipher modes, cipher block chaining.**

I. Introduction

The focus of this paper is on giving a theory and automated techniques for discovering common types of known-pair and chosen-text attacks for block ciphers. The objective of attacks on cryptosystems have historically been to recover plaintext from ciphertext, or to deduce the decryption key. However, the objective may also be to defeat message and session integrity based on such cryptosystems.

A primary attack to defeat a cryptosystem or the integrity of a protocol session is to mount a chosen-plaintext attack. A *chosen-plaintext attack* is one where the adversary chooses plaintext and is then given the corresponding ciphertext. Using the chosen-plaintext attack, the adversary can perform a “dictionary attack” to learn the key. To perform a dictionary attack, the adversary stores in a table the encryption of the same plaintext under numerous keys. When the real ciphertext is given, the adversary looks in a table to see what key corresponds to the given ciphertext. A chosen-plaintext attack also can enable the adversary to exercise the cryptosystem to create illegitimate messages that may be accepted as authentic. A *chosen-ciphertext attack* is one where the adversary chooses ciphertext and is then given the corresponding plaintext. A protocol subject to a chosen-ciphertext attack allows the adversary to read encrypted messages. A *known-pair attack* is characterized by an adversary learning a quantity of plaintext and corresponding ciphertext. On the surface, it can be surprising that a known-pair attack is significant given large message spaces and key sizes. However, the security of some protocols depend on the absence of any known plaintext-ciphertext pairs. For example, in [10], the discovery of a known pair in the Privacy Enhanced Message protocol (PEM) led to the construction of an attack having success probability of 100%. Consequently, it is important to be able to verify assumptions that no such pairs are known by those other than the intended entities. As in the case of PEM, analysis of even a seemingly simple secure email protocol becomes difficult. Herein lies the motivation of this work.

Guidance on how to avoid chosen-text and known-pair attacks for block ciphers has been in the literature for many years [14]. However, attacks based on chosen-text and known-pairs continue to be a problem even in security protocols designed and reviewed by teams of security experts. In spite of this, little work has been done on developing formal theories and automated techniques for detection and prevention of chosen-text and known pair attacks. This is mainly because the degree of abstraction at which the problem occurs is at a somewhat lower level than that usually handled by most of the existing applications of formal methods of cryptographic protocols. Furthermore, analysis is tedious and complex. Typically there are many details to consider including numerous message types and protocol options during the lifetime of keys. Consequently, the search space can become very large. However, as the work of Stubblebine and Gligor [11], [12] has shown us, it is possible to apply formal techniques even at the level at which splicing and message decomposition occur and obtain meaningful results.

The paper is organized as follows. In Section II, we use a state machine model for giving a formal characterization of known pairs and chosen text in the context of a cryptographic protocol. In Section III, we justify how a semiautomatic technique using the NRL Protocol Analyzer [6] corresponds to our characterization of known pairs and chosen text. In Section IV, we an perform an example analysis of the IP encapsulating security payload (ESP) protocol [1]. Here we rediscover known attacks, and discover new variants of known attacks. In Section V, we conclude. In Appendix A, we give the full specification of the ESP Protocol, while in Appendix B we give the actual queries we presented to the NRL Protocol Analyzer.

Stuart Stubblebine is with CertCo, 55 Broad St. - Suite 22, New York, NY 10004, USA. Email: stubblebine@{certco.com, cs.columbia.edu}, http://www.cs.columbia.edu/~stu . Work by this author was also performed at AT&T Research.

Catherine Meadows is with Naval Research Laboratory, Code 5543, Washington DC 20375, USA. Email: meadows@itd.nrl.navy.mil

II. the Γ Model

In this section we use a state machine model to formally express the security properties of known pairs and chosen text. We refer to this model throughout the rest of the paper as the Γ model. The protocol properties of known-pair and chosen-text are expressed as secure state invariants. Protocol actions are expressed as state transition rules. In Section III, the model is used to justify the correctness of our procedure to use tools for automated analysis.

By convention, superscripts are used to identify principals, and subscripts identify the particular state. Where the meaning is unambiguous we sometimes leave out certain subscripts and superscripts.

A. Definitions

Messages. Let $\mathbf{Z}$ be the set of all strings of zeros and ones of finite length. The set of messages is a subset of $\mathbf{Z}$ defined as follows. Underlying the set of possible messages is the set $\mathcal{T}$ of *primitive terms*. $\mathcal{T}$ can be of different *types.* One type is *keys.* Other possible types are principal identifiers, random numbers, and sequences numbers, etc. Types may be thought of as nonintersecting subsets of $\mathbf{Z}$. Messages are defined by mutual induction. We first define a number of functions, including encryption, decryption, exclusive-or, concatenation, and deconcatenation, from subsets of $\mathbf{Z}$ to $\mathbf{Z}$.

Given a set of terms $\mathcal{V}$, we recursively define the set of messages $\mathcal{M}(\mathcal{V})$ as follows.

- B1. *Primitive terms.* $X \in \mathcal{M}(\mathcal{V})$ if $X \in \mathcal{V}$.
- B2. *Block encryption.* $e_k(X) \in \mathcal{M}(\mathcal{V})$ if $X \in \mathcal{M}(\mathcal{V})$ and $k \in \mathcal{V}$ and k is of type *keys.*
- B3. *Block decryption.* $e_k^{-1}(X) \in \mathcal{M}(\mathcal{V})$ if $X \in \mathcal{M}(\mathcal{V})$ and $k \in \mathcal{V}$ and k is of type *keys.*
- B4. *Exclusive-Or.* $X \oplus Y \in \mathcal{M}(\mathcal{V})$ if X and $Y \in \mathcal{M}(\mathcal{V})$.
- B5. *Concatenation.* $(X_1, X_2) \in \mathcal{M}(\mathcal{V})$ if X_1 and $X_2 \in \mathcal{M}(\mathcal{V})$.
- B6. *Deconcatenation.* X_1 and $X_2 \in \mathcal{M}(\mathcal{V})$, if $(X_1, X_2) \in \mathcal{M}(\mathcal{V})$.

Given a set of terms S such that $S \subset \mathcal{M}(\mathcal{T})$, we define $\mathcal{M}(S)$ to be the set of messages computable from S recursively.

Note that we have not yet explicitly defined any of the functions we have used. As a matter of fact, we will not define them completely. However, we will assume that they obey a collection of algebraic identities: namely, that encryption and decryption with the same keys cancel each other out, that exclusive-or is commutative, associative, possesses an identity, and is self-cancelling, and that concatenation and deconcatenation cancel each other out. These identities will later be used to construct a correspondence with the NRL Protocol Analyzer model.

Using the definition of block encryption, we can formally define plaintext-ciphertext pairs. (We will use the term *pair* to refer to a plaintext-ciphertext pair.)

Definition 1: A *plaintext-ciphertext pair* is an ordered 2-tuple $\{\langle p, c\rangle \mid \exists k \text{ s.t. } c = e_k(p) \wedge p = e_k^{-1}(c)\}$.

Definition 2: A *plaintext-ciphertext pair under key, k,* is an ordered 3-tuple $\{\langle p, c, k\rangle \mid c = e_k(p) \wedge p = e_k^{-1}(c)\}$.

We can now define the Cipher Block Chaining Mode of encryption in terms of block encryption and decryption. In the analysis section, we use this definition to specify specify details of the message format for an Internet security protocol. Other modes of decryption can be specified in a straight-forward manner using our definitions of block encryption and exclusive-or.

Definition 3: **CBC Mode**

1. Encryption.

Input: m-bit key k; l-bit IV; l-bit plaintext blocks $p_1, \dots p_r$.

Output: $c_0, \dots, c_r$ such that $c_0 \leftarrow IV$ and $c_i \leftarrow e_k(c_{i-1} \oplus p_i)$ for $1 \le i \le r$.

2. Decryption.

Input: m-bit key k; l-bit IV; l-bit ciphertext blocks $c_1, \dots c_r$.

Output: $p_0, \dots, p_r$ such that $p_0 \leftarrow IV$ and $p_i \leftarrow c_{i-1} \oplus e_k^{-1}(c_i)$ for $1 \le i \le r$.

B. Principals and States

Our system entities are called *principals.* A distinguished principal is called the *intruder.* Typically, the intruder is assumed to be able to observe message communications and insert messages. The intruder may also share state information and direct actions with one or more *dishonest principals.* Thus, to simplify our model, the intruder includes dishonest principals. That is, we do not explicitly distinguish between the intruder and dishonest principals.

Definition 4: A *local state* for principal i is the tuple $L^i = (Know^i)$ where $Know^i$ is the set of messages known by i.

Definition 5: A *global state* is the tuple $\mathcal{S} = \{L^0, L^1, \dots, L^n\}$ consisting of the intruder state L^0 and all local states.

Definition 6: The *initial state* is the tuple $\mathcal{S}_0 = \{L^0, L^1, \dots, L^n\}$ where each local state, i, is $(Know^i = \{\emptyset\})$.

The addition of 'initial' keys and knowledge of 'initial' messages is specified as an explicit step of a protocol run.

C. State Transition Rules and Run

There are three state transition rules: send, receive, and choose. All principals, including the intruder, may perform the first two. Only the intruder may perform the last.

Principal Operations. State transitions occur due to the following operations. In what follows, I represents the intruder and R represents an arbitrary principal. $Know'$ represents the updated state variable $Know$.

$$send^R(message:\ message \in Know^R)$$
$$Know'^I \leftarrow \mathcal{M}(message \cup Know^I)$$

A principal can send a message it knows. Sending a message has the effect of adding the message to the *intruder's* knowledge.

$$receive^R(message)$$
$$Know'^R \leftarrow \mathcal{M}(message \cup Know^R)$$

Receiving a message has the effect of adding the message to the principal's knowledge.

We also define the notion of choosing, or creating, a message nondeterministically.

$choose^{R}(message: \ message \in_{nd} [0,1]^{n})$
$\quad Know'^{R} \leftarrow \mathcal{M}(message \cup Know^{R})$

For this analysis, we are assuming that only the intruder can perform choose. Other principals may create random messages, but we assume that these are known initially. The main point of the choose transition is to provide help in modelling of chosen plaintext and ciphertext attacks.

Processes and Run. Each principal consists of a communicating process that performs principal operations that update the state. A *Run* is a sequence of global states $\mathcal{S}_0, \ldots, \mathcal{S}_m$.

D. Secure State Invariants

We express security properties of known and chosen pairs as state invariants of cryptographic protocols. Implicit to our invariants is that the intruder I is able to relate corresponding blocks of plaintext and ciphertext that she knows. This is reasonable where the protocol specification is known to the intruder.

Invariant 1: A global state, $\mathcal{S}_j$, is **known-pair secure** with respect to a principal, I, and a key, k, iff

$\langle p, c, k\rangle \ \supset \neg(p \in (Know_j^I) \ \wedge \ c \in (Know_j^I))$.

There are no known pairs under the key k if *both* plaintext and the corresponding ciphertext for a pair are not known to the intruder.

Invariant 2: A global state, $\mathcal{S}_j$, is **chosen-plaintext secure** with respect to a principal, I, and a key, k, iff for $0 \leq m \leq j-1$,

$choose_m^I(p) \supset \neg(e_k(p) \in (Know_j^I))$.

A principal choosing plaintext in a prior state does not result in the intruder knowing the corresponding ciphertext in the current state. Typically, this property should hold for the entire run (for all possible runs). Chosen-plaintext attacks can lead to vulnerabilities that enable the adversary to forge messages using the key, read messages, or learn the key. Messages can be forged by exercising the system to create new messages. The key can be learned by precomputing a dictionary of a chosen plaintext under the key space. Note, however, that in all these cases it must be possible for the adversary to obtain the corresponding ciphertext.

Invariant 3: A global state, $\mathcal{S}_j$, is **chosen-ciphertext secure** with respect to a principal, I, and a key, k, iff for $0 \leq m \leq j-1$,

$choose_m^I(c) \supset \neg(e_k^{-1}(c) \in (Know_j^I))$.

A principal choosing ciphertext in a prior state does not result in the intruder knowing the corresponding plaintext in the current state. Typically, this property should hold for the entire run (for all possible runs).

III. Automating the Analysis

We begin by giving a brief introduction to the NRL Protocol Analyzer. We then compare the rules of the analyzer with the rules outlined in the previous section. We go on to augment the analyzer rules to capture the necessary formalism. We give informal arguments on the mapping between our definitions and analysis using the analyzer.

A. Overview of the Analyzer

The NRL Protocol Analyzer is a formal methods tool for analyzing security properties of cryptographic protocols. Protocols are specified as communicating state machines, one of which is a hostile intruder who can read all traffic, modify or delete traffic, perform cryptographic operations, and may be in cooperation with some legitimate users of the system. The intruder also is assumed to have knowledge about relationships between data. That is, if it sees the result of encrypting a piece of plaintext with a key, it will know that it is the result of encrypting the plaintext with that key, although it may not know plaintext or key. This model has the effect of simplifying our search for known pairs, since it means that if the intruder sees a plaintext-ciphertext pair, it also knows that it is such.

Each honest participant is represented as one of these machines, each of which possesses a set of local state variables. These local state variables are particular to the particular protocol specification.

A participant is defined with respect a party, plus a rule, plus the particular execution of the protocol.

The user of the Analyzer attempts to determine whether or not a protocol is secure by specifying an insecure state. The Analyzer works backwards from that state until it either reaches an initial state, in which case it has found an attack, or until it has shown that all paths begin in an unreachable state.

The Analyzer makes no assumptions about the limits on the number of protocol executions, the number of principals performing the different executions, the number of interleaved executions, or the number of times cryptographic functions are applied. This results in a search space that is originally infinite. However, the Analyzer provides means for specifying and proving inductive lemmas about the unreachability of infinite classes of states. This allows the user to narrow down the search space so that in many cases an exhaustive search is possible.

An Analyzer specification consists of four parts. The first part describes the operations and terms used in the specification. The second part describes the rewrite rules used. These capture the necessary algebraic properties of cryptosystems, such as the fact that encryption and decryption cancel each other out. The third part specifies the words known initially by the intruder. The last and main part consists of transition rules describing the actions taken by legitimate participants. The inputs to these transition rules are messages received (which may have been passed on by the intruder) and values of local state variables. The outputs are messages sent (which will be available to the intruder) and new values of the local state variables. The

intruder actions are not specified directly, but are generated automatically from the specification.

The intruder actions are operations such as encryption, decryption, concatenation, etc., according to the specification. An intruder action is modeled in the Analyzer by the intruder sending a message to itself, where the message is constructed by applying an operation to a word known by the intruder.

This allows us to define the local state for intruders and principals as follows.

Definition 7: A *local state* for the intruder 0 is the set $L^0 = (\mathcal{W})$ where $\mathcal{W}$ is the set of messages previously sent, including messages sent by the intruder to itself. A *local state* for an honest principal is the set of values for honest principal's local state variables. The global state is the collection of all local states.

The Analyzer generates the input state to an existing global state as follows. It takes each transition rule output in turn and uses a narrowing algorithm to find a complete description of all substitutions that make any portion of the output reducible to a subset of the state. The input to the rule and the unmatched portion of the state then become the input state. This allows the Analyzer to discover unexpected consequences of the use of cryptographic functions.

The Analyzer has been applied to a number of different cryptographic protocols, and has found flaws in several. In some cases the flaws had not been discovered before. Examples of protocols the Analyzer has been used to examine are the Simmons Selective Broadcast Protocol [5], the Burns-Mitchell Ticket Granting Protocol [4], the Needham Schroeder public key protocol [8], and, most recently, the Internet Key Exchange protocol [7]. A description of the Analyzer is given in [6].

Most of the protocols to which the Analyzer has been applied are specified at a higher degree of abstraction than the protocols we will be looking at in this paper. Methods for using block ciphers to encrypt messages more than one block long are not modeled, and the integrity of an encrypted message is assumed. Thus, using the Protocol Analyzer to explore the consequences of cipher block chaining was a new departure.

B. Correspondence Between the Formal Models

We are unable to show that properties of "known-pair secure", "chosen-plaintext secure", "chosen-ciphertext secure" are satisfied by certain queries. As mentioned previously, this is because the Analyzer is limited in handling commutativity and associativity properties of exclusive-or. However, we strive for a lesser, but useful goal of informally explaining the correspondence between protocol-independent analyzer specifications and the formal model of Section II. This informal argument will support that claim that a proof of security in our formal model implies a proof of security in the Analyzer model, although it will not support claims in the other direction. However, eventually, it may become practical to overcome the limitations of handling commutativity and associativity properties of exclusive-or in the Analyzer. The discussion here is a first step towards a correspondence proof showing that protection properties are actually satisfied given results using the tools.

There are three main differences between the Analyzer model and the Γ model. One is that a local state in the Analyzer model, instead of consisting of all words computable from a set of received messages and created words, instead consists of a set of words explicitly computed from that set. The other is that the set of possible identities on words is restricted to those that can be written as Noetherian, confluent rewrite rules. The third is that, while the Analyzer model is defined in terms of symbols that may be related by reduction rules, so that, for example, a term `e(X,Y)` is not assumed to be equal to any other symbol unless it is explicitly equivalent to it modulo the defined reduction rules. The Γ model, however, is defined in terms of actual functions, which although they also obey algebraic properties, are maps from one string of bits to another instead of symbols.

Our solution to the problem of moving from functions to symbols is to replace the Γ model by an interim model, the Γ' model, in which the evaluation of functions is replaced by a symbol defining the function. Thus, if X and k are strings of bits, then $e_k(X)$, is an uninterpreted symbol instead of another string of bits. We say that two symbols in the Γ' model are *equivalent* if they are equivalent under the assumptions that concatenation and deconcatenation cancel each other out and vice versa, that encryption and decryption cancel each other out and vice versa, and that exclusive-or is commutative and associative, possesses an identity, and is self-cancelling. Finally, the choose operation is replaced by adding the term notsent(W) to the intruder's knowledge set, where W is a free variable.

We will now show that there is a natural map from the Γ' to the Γ model taking each symbol to the result of evaluating the functions used in that symbol, and that any run in the Γ' model maps to a run in the Γ model, but not necessarily the other way around. In the remainder of this section we will refer to the Γ' model exclusively.

In the case of operations, we were able to obtain a one-to-one correspondence between the NRL model and the Γ' model. The operations we used were as follows. First, the Analyzer offered the following built-in operations:

a. Given X and Y, produce (X,Y), and

b. Given (X,Y), produce X and Y.

For purpose of introducing a correspondence with the Γ', we also included the following operations:

```
op(1):e(key(U,V),X) --> W:::pen.
op(2):d(key(U,V),X) --> W:::pen.
op(3):exor(X,Y) --> W:::pen.
```

The operation "e" refers to encryption, "d" to decryption, and "exor" to exclusive-or.

We also used the following rewrite rules.

```
rr(1):e(X,d(X,Y)) => Y.
rr(2):d(X,e(X,Y)) => Y.
rr(3):exor(X,exor(X,Y)) => Y
```

Thus we did not include the associative-commutative properties of exclusive-or due to the limitations of the narrowing algorithm used by the Analyzer.

We will make use of the following lemma:

Lemma: Let W be a set of irreducible words in the Analyzer model. Then there is a natural map i from W onto a set of words in the Γ' model defined by

a. $\mathrm{i}(\mathrm{e}(X,Y)) = e_{i(X)}(i(y))$;
b. $\mathrm{i}(\mathrm{d}(X,Y)) = e^{-1}_{i(X)}(i(y))$;
c. $\mathrm{i}(\mathrm{exor}(X,Y)) = i(X) \oplus i(Y)$;
d. $\mathrm{i}((X,Y)) = (\mathrm{i}(X),\mathrm{i}(Y))$;
e. $\mathrm{i}(\mathrm{notsent}(X)) = \mathrm{notsent}(\mathrm{i}(X))$;

and where i maps words of a given type in the Analyzer model to words of the same type in the Γ' model, and where i is the identity on free variables.

Proof: All we need to show is that i is well-defined. This follows directly from that fact i maps the set of rewrite rules in the Analyzer model to a subset of the set of relations in the Γ' model.

Using the above lemma, we can use i to induce a map from any set of W of irreducible words in the Analyzer model to a subset of $\mathcal{M}(i(W))$.

Theorem : There exists a well-defined map from any set of paths produced by the Analyzer to a set of paths in the Γ' model.

Proof: Let P be a path in the Analyzer model. In other words, P is a sequence of global states, in which the transition from one state to another is accomplished by some combination, for a particular principal A, of the sending of a message by A that was produced by applying operations to A's local state (hence adding it to the intruder's local state), the receiving of a message by A, and the adding of words to A's local state. We map this path to a path in the Γ model, denoted by $\mathrm{i}(P)$, by first replacing each local state S_R with $\mathcal{M}(i(S))$ and each message M with $\mathrm{i}(M)$ and removing any trivial state transitions. This does not quite give us a path in the Γ' model, because in our new run we assume that all words of the form choose(W) are known initially. We can turn it into a path, however, by replacing that initial state by a series of states and transitions where the initially known words and the words chosen by the intruder are produced.

C. Queries for Finding Known and Chosen Pairs

Claim 1: If a search for a state in which the the intruder knows `X` and `e(key(U,V), X)` has a solution, then the protocol is not known-pair secure for some run with respect to the intruder and `key(U,V)`.

This follows directly from the definitions and the correspondence between the formal models. As illustrated in the following section, the analysis method identifies the run for which the protocol fails.

Our queries for chosen plaintext and ciphertext make use of the notsent operator, as follows:

Claim 2: If a search for a state in which the the intruder knows `e(key(U,V),notsent(W))` has a solution, then the protocol is not chosen-plaintext secure for some run with respect to the intruder and `key(U,V)`.

Claim 3: If a search for the intruder knowing `d(key(U,V),notsent(Z))`, has a solution, then the protocol is not chosen-ciphertext secure for some run with respect to the intruder and `key(U,V)`.

The above two claims follow from the fact that paths producing the word `d(key(U,V),notsent(Z))` or the word `e(key(U,V),notsent(W))` map to paths producing the same words in the Γ' model. These in turn map to paths producing the words $e_{K_1}(Z_1)$; $e^{-1}_{K_2}(Z_2)$ in the Γ model, where Z_1 and Z_2 were produced using the choice transition.

IV. Analysis of the IP Encapsulating Security Protocol

In this section we describe our analysis of an early version of the IP Encapsulating Security Protocol or ESP [1], [9]. ESP is the protocol of the Internet security architecture for securing IP [2]. It describes formats and transforms for encryption and decryption of data. We selected ESP because its use of CBC mode caused Bellovin to notice a number of possible attacks when it was used under certain system configurations [1], [9].

A. Description of the Attacks

In Bellovin's scenario, secure communication is between hosts. Each pair of hosts shares a key, which is not changed between sessions. Each host has a number of users communicating with it, some of whom are honest, and some of whom are actively trying to subvert the protocol. Cipher block chaining is used to encrypt packets, and IVs are sent in the clear. Each packet contains an unencrypted ESP header, containing a SPI (Security Parameter Index) which corresponds to a key shared between hosts. SPIs and their corresponding keys are one-way, that is, if host A initiates contact with host B, it uses a different SPI than if host B initiates contact with host A. Encrypted packets contain headers that include such information as to who sent the message and for whom it is intended. Depending upon the communication protocol used, different types of header formats may be sent.

There are at least two ways in which an intruder could subvert the protocol:

a. It could pass a message from itself as coming from an honest user U by appending a fake header H1 identifying the message as coming from U to its beginning and giving it to the host on which U resides to encrypt. The host would append another header H2 to the message and encrypt it. The intruder could then truncate the encrypted message at H1, and pass it off as a message with header H1, using the last block of the encrypted H2 as the IV. Bellovin describes a similar attack in [3].
b. If the intruder resides at host A, it could learn a message M1 intended for another user U at the same host A as follows. First it would need to find out the host

B originating the message. It would take another message M2 from B to A intended for herself, and remove the encrypted header EH2. She would then append the last portion M1' of M1 to EH, so that it is of the expected length. The host would then decrypt the entire message, and return the decrypted M1' to the intruder. Part of M1' could be garbled, but thanks to the self-healing properties of cipher block chaining, the remainder would be readable. Bellovin describes this attack in [3].

B. The Specification

In order to see whether the Analyzer could find attacks like this, we modeled a protocol along the line of Bellovin's scenario. In order to make the analysis tractable, we assumed packets were only four blocks long. We modeled two types of headers. One was two blocks long, and one was only one block. This meant that message bodies were always at least two blocks long, which meant that we were able to illustrate the self-healing properties of cipher block chaining; even if the first block was garbled, the second would be correct. We also assumed that hosts would only continue decrypting a packet if the header parsed; this allowed us to discard cases in which header was garbled, and hence a message could neither be passed to the intruder or regarded as a legitimate message for an honest user. This allowed us to ignore a number of useless states that would not have aided us in the analysis but would have increased the size of the search space. We also assumed that SPIs were simplex, so that host A encrypting a message for host B always used `spi(A,B)` when encrypting a message, and `spi(B,A)` when decrypting it. A similar assumption was made about keys.

We made a severely simplifying assumption about headers. We assumed that the only thing distinguishing one header from another was its type (short or long) and who the senders and receivers of the message were. This was probably oversimplification, and may have prevented us from finding some subtle attacks in which different sessions between the same parties were confused. However, we believed that it was enough for our goal of determining whether sessions between different parties could be confused. We also assumed that the intruder knows all headers.

We specified cipher-block chaining in four types of messages. Two described the encryption of short and long header messages, respectively for an honest principal, and two described the encryption of short and long header messages for dishonest principals. The main difference between the two types of messages was that for honest principals, the words encrypted were or the form

$$\texttt{mess}(\texttt{user}(\texttt{A},\texttt{honest}),\texttt{user}(\texttt{D},\texttt{H}),\texttt{ts}(\texttt{host}(\texttt{C}),\texttt{M},\texttt{N}),\texttt{X}),$$

representing a message from honest `user(A,honest)` to another principal `user(D,H)` encrypted by host `host(C)`. For dishonest principals (assumed to be in league with the intruder) the words are supplies by the intruder. In all cases the encrypted messages are learned by the intruder.

Our specification of decryption is similar, except that in all cases the messages to be encrypted are supplied by the intruder, while the decrypted messages are only returned to the intruder when the principal for which it is being decrypted in dishonest.

A copy of the specification is included in Appendix A.

C. Results of Using the Analyzer

We posed four questions to the Analyzer. In the first two, we asked it whether or not if could find

$$\texttt{e}(\texttt{key}(\texttt{host}(\texttt{A}),\texttt{host}(\texttt{B})),\texttt{notsent}(\texttt{X}))$$

(chosen plaintext), and

$$\texttt{d}(\texttt{key}(\texttt{host}(\texttt{A}),\texttt{host}(\texttt{B})),\texttt{notsent}(\texttt{X}))$$

(chosen ciphertext). For chosen plaintext, the Analyzer found the word

$$\texttt{e}(\texttt{key}(\texttt{host}(\texttt{A}),\texttt{host}(\texttt{B})),\texttt{notsent}(\texttt{X}))$$

unreachable in a few seconds. For chosen ciphertext, the Analyzer produced an infinite number of paths containing arbitrarily long cipher chains, but the shortest paths were only three state transitions long. An example of a typical one follows. Since a state transition on the Analyzer often covers several steps in an attack as we present it, we annotate each step, in this and subsequent attacks, with the transition in which it occurred.

1. (Transition 1) Host A encrypts message from honest user U at to dishonest user V at Host B.
2. (Transition 2) V sends IV and encrypted message header EH1 and EH2 from the message in [1] on to B.
3. (Transition 2) V appends notsent(W) as the next part of the encrypted message. This is also sent to B.
4. (Transition 2) B decrypts message, and returns exor(EH2,d(key(host(A),host(B)),notsent(W))) to the dishonest user. This is seen by the intruder, since all messages sent to dishonest users are shared with the intruder.
5. (Transition 3) The intruder of course has already learned EH2 from Step 2. He uses it to find d(key(host(A),host(B)),notsent(W)) from the message sent in step [4].

The Analyzer also found a number of attacks similar to Bellovin's. We asked it two questions (described in more detail in Appendix B). The first was an integrity question which asked whether the first block of a decrypted message stored in the host could be a header indicating it had been sent by an honest principal, while the second block was notsent(W). This search took just a few seconds, and generated several attacks. Among the paths returned was the following:

1. (Transition 1) Host A encrypts a message M1 from dishonest user V to some user U1 at host B. The first part of the message is a header for a message M2 from an honest user U2. The rest is the spoofed message

(e.g. `notsent(W)` followed by an arbitrary block H) the intruder wants to send. After intercepting the encrypted message, the intruder removes the encrypted message header.

2. (Transition 2) V sends the three encrypted blocks to B, passing off the first block as the IV. A third and fourth random block are included to make up the entire packet; these are indicated by variable words in the Analyzer.
3. (Transition 2) B decrypts the apparent first part of the message, and accepts that as the header.
4. (Transition 2) B decrypts the apparent rest of the message, and accepts that as the message from E.

Bellovin describes a related but different attack in [3]. In his attack a legitimate message M2 is sent from U1 to U2. The intruder constructs a message M1, and then cuts off the portions after the headers on M1 and M2. The last portion of M1 is appended to the first part of M2. This attack is somewhat stronger than ours, since it allows an intruder to hijack a session without necessarily knowing the header associated with that session.

We next attempted to find an unauthorized disclosure attack. We produced this by representing a block of a message produced to be sent from `user(U1,honest)` at host A to `user(U2,honest)` at host B as `message(user(U1,honest),user(U2,honest), ts(host(A),M,N),Num)` where `Num` indicates the block's position in the packet and `ts(host(A),M,N)` is a host timestamp guaranteeing the uniqueness of the message. This time the Analyzer again produced an infinite number of attacks (or would have if we had let it run forever) involving arbitrarily long cipher chains, but the shortest attack involved only three transitions, as follows:

1. (Transition 1) Honest user U1 at Host A encrypts a message to another honest user U2 at Host B. The result of this is IV1, EncryptedHeader1, Message1.
2. (Transition 2) Another user U3 at Host A encrypts a message to dishonest user V at Host B. In this case, all we are interested in is the header. The result of this is IV2, Encryptedheader2.
3. (Transition 3) The intruder sends the message IV2, EncryptedHeader2, EncryptedHeader1, Message1 to Host B.
4. (Transition 3) Host B decrypts the header, and concludes that the remainder of the message will be for the dishonest user V. It then uses EncryptedHeader 2 to decrypt EncryptedHeader1, which is garbage. It then uses EncryptedHeader1 to decrypt Message1, which is passed to the intruder.

Note that the garbage decryption of EncryptedHeader1 is also passed on to the intruder, but, since this does not aid in the attack, it does not appear in the search.

This is exactly the unauthorized disclosure attack found by Bellovin.

V. Conclusion

This work constitutes a significant step towards proving secure uses of mechanisms that assume absence of known or chosen pairs. We have given a characterization of known-pair and chosen-text security, and a means of specifying their presence or absence. We have also shown how it can be applied to the use of an existing tool, the NRL Protocol Analyzer, and how the tool could use the model to find ways of generating chosen pairs in the ESP protocol. We then went on to show how it could be extended to be used to find attacks based on cutting and pasting.

Although we have shown that our theory and automated analysis technique is successful in finding attacks, additional work can be done to streamline the process. In particular, we might consider strategies for developing specifications for the Analyzer. The assumptions we made (that all packets were four blocks long, that headers took up one or two blocks entirely) were designed with the idea in mind of capturing the relevant security features of the specification, while still keeping the analysis tractable. However, our approach to doing this was informal and and somewhat ad hoc. We might develop more systematic ways of identifying assumptions, as well as rigorous arguments that our abstractions capture the assumptions. This will give us ways of generating specifications in a more systematic fashion, and will allow us more confidence in the usefulness of any proofs that we obtain that a specification is secure against certain types of attacks. However, as we learned from writing the specification that appears in this paper, it is possible to clarify assumptions and discover errors by examining the attacks produced by the Analyzer and comparing them against what we know to be possible. We expect continued use of the Analyzer to test our hypotheses to help us further in the development of our heuristics.

Finally, there are probably some improvements that need to be made to the Analyzer itself to lend itself to analysis of more complex protocols involving low-level properties. Surprisingly, we did not find the Analyzer's inability to model the commutativity and associativity of exclusive-or as much of a handicap as we had feared. This was because many threats posed by cipher block chaining, that is, its cut-and-paste and self-healing properties, were easily expressible in terms that could be modeled by the Analyzer. However, it did reduce the effectiveness of the Analyzer in showing that such attacks were not possible, and clearly it is necessary to extend the capacities of the Analyzer in this direction.

Another problem was the state explosion as a result of the generation of an infinite series of arbitrarily long attacks. We believe that the complexity of the specification could be simplified by allowing specifications to be defined in a more recursive manner. The actual operations that are involved in cipher block chaining are simple; what was complex was describing their iteration. A recursive style of specification would make this task easier. Indeed, an earlier specification we had done for the ESP protocol [13] was recursive, and much simpler. However, we found that it introduced an error, since it allowed an intruder to learn as output the first ciphertext block of a message before the next block had been input. This produced a bogus chosen-plaintext attack. Unfortunately, there is no other way that

the current version of the Analyzer could be made to support recursion in this case. This was because, although we could use the message space as a buffer that could hold an arbitrarily large number of messages, we could not do the same for state variables. Since we are interested in supporting recursive specification for other purposes as well (for example, for dealing with group protocols), a modification of the Analyzer language so that it supports recursion more flexibly by allowing state variables to act as arbitrary-sized buffers seems to be in order.

The problem of state explosion is tougher. However, we note that it was actually very similar to the type of state explosion that motivated the development of inductive lemmas for limiting the Analyzer search space[6]. These lemmas involve generating languages such that an intruder can't learn a word in a language unless it already knows a word in the language. This approach would not work by itself for the infinite sequences we generated in the ESP analysis: the languages are based on the fact that every path to a state is infinite, while in the infinite cases we discovered in the ESP case, only some paths were infinite, while others are finite and begin in an initial state. Thus some additional (but possibly similar) mechanisms will have to be introduced to handle these kinds of cases.

References

[1] R. Atkinson. IP encapsulating security payload (ESP). Request for Comments (Proposed Standard) RFC 1827, Internet Engineering Task Force, August 1995.

[2] R. Atkinson. Security architecture for the internet protocol. Request for Comments (Proposed Standard) RFC 1825, Internet Engineering Task Force, August 1995.

[3] S. M. Bellovin, Problem Areas for the IP Security Protocols, *Proceedings of the Sixth Usenix UNIX Security Symposium*, San Jose, CA, July, 1996.

[4] C. Meadows, A System for the Specification and Verification of Key Management Protocols, *Proceedings of the 1991 IEEE Computer Society Symposium in Research in Security and Privacy*, May 1991.

[5] C. Meadows, Applying Formal Methods to the Analysis of a Key Management Protocol, *The Journal of Computer Security*, 1992:1,1, Jan, 1992.

[6] C. Meadows, The NRL Protocol Analyzer: An Overview, *J. Logic Programming*, 1996:19, 20:1-679, Elsevier Science, New York, NY, 1996.

[7] C. Meadows, Analysis of the Internet Key Exchange Protocol Using the NRL Protocol Analyzer, *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, May 1999.

[8] Catherine Meadows, Analyzing the Needham-Schroeder Public-Key Protocol: A Comparison of Two Approaches, *Computer Security - ESORICS 96*, Springer-Verlag, September 1996, pp. 365-364.

[9] P. Metzger, P. Karn, and W. Simpson. The ESP DES-CBC transform. Request for Comments (Proposed Standard) RFC 1829, Internet Engineering Task Force, August 1995.

[10] S. Stubblebine and V. Gligor, Protecting the Integrity of Privacy-Enhanced Electronic Mail with DES-Based Authentication Codes, PSRG Workshop on Network and Distributed System Security, San Diego, CA, Feb. 1993.

[11] S. Stubblebine and V. Gligor, Protocol Design for Integrity Protection, *IEEE Computer Society Symposium on Research in Security and Privacy*, Oakland, CA, May, 1993, pp. 41-53.

[12] S. Stubblebine and V. Gligor, On Message Integrity in Cryptographic Protocols, *IEEE Computer Society Symposium on Research in Security and Privacy*, Oakland, CA, May, 1992, pp. 85-104.

[13] S. Stubblebine and C. Meadows, On Known and Chosen Cipher Pairs, *Proceedings of DIMACS Workshop on Design and Formal Verification of Security Protocols*, Rutgers, New Jersey, Sept. 1997, `http://dimacs.rutgers.edu/Workshops/Security/program2/program.html`.

[14] V. L. Voydock and S. T. Kent, Security Mechanisms In High-Level Network Protocols, *Computing Surveys*, vol. 15, no. 2, June 1983.

Appendix A: Specification of the ESP Protocol

```
fs(1):host(C) --> W::.
fs(2):iv(TS) --> W::.
fs(3):key(U,V) --> W::.
fs(4):header(U,V,Type,Num) --> W::.
fs(6):mess(U,V,TS,Num) --> W::.
fs(7):plus(Num) --> W::.
fs(8):ts(U,M,N) --> W::.
fs(9):user(A,H) --> W::.
fs(10):spi(U,V) --> W::.
fs(11):notsent(X) --> W::.

op(1):e(K,X) --> W:::pen.
op(2):d(K,X) --> W:::pen.
op(3):exor(X,Y) --> W:::pen.

atom(1):honest --> W:.
atom(2):dishonest --> W:.
atom(3):nil --> W:.
atom(4):one --> W:.
atom(5):short --> W:.
atom(6):long --> W:.
atom(7):word --> W:.
atom(8):header --> W:.
atom(9):two --> W:.
atom(10):three --> W:.
atom(11):initial123 --> W:.
atom(12):ball123 --> W:.
atom(13):noball123 --> W:.

rr(1):e(X,d(X,Y)) => Y.
rr(2):d(X,e(X,Y)) => Y.
rr(3):exor(X,exor(X,Y)) => Y.

known: host(C), user(A,H), header(U,V,Type,Num),
mess(user(C,dishonest),V,TS,Num),
notsent, spi(H1,H2),
plus(Num), notsent(X).

Session host_main(host(C),N,N):
host_honestlongencrypt,
Or:
host_honestshortencrypt,
Or:
host_badguyshortencrypt,
Or:
host_badguylongencrypt,
Or:
host_badguylongdecrypt,
Or:
host_badguyshortdecrypt,
```

```
Or:
host_goodguylongdecrypt,
Or:
host_goodguyshortdecrypt.

Subroutine
  host_goodguyshortdecrypt(host(B),M,N) :
If:
rcv msg(user(A,H),host(B),
 [spi(host(C),host(B)),IV,Z1,Z2,Z3,Z4],M):
verify(exor(IV,d(key(host(C),host(B)),Z1)),
   header(user(A,H),user(D,honest),
    short,one)):
Then:
host_block1 :=
  exor(IV,d(key(host(C),host(B)),Z1)),
host_block2 :=
  exor(Z1,d(key(host(C),host(B)),Z2)),
host_block3 :=
  exor(Z2,d(key(host(C),host(B)),Z3)),
host_block4 :=
  exor(Z3,d(key(host(C),host(B)),Z4)):
event(host(C),
  [host(B),user(A,H),user(D,honest)],
  host_goodguyshortdecrypt,
  [iv(ts(host(C),M,N))],M).

Subroutine
  host_goodguylongdecrypt(host(B),M,N) :
If:
rcv msg(user(A,H),host(B),
 [spi(host(C),host(B)),IV,Z1,Z2,Z3,Z4],M):
verify(exor(IV,d(key(host(C),host(B)),Z1)),
   header(user(A,H),user(D,honest),
     long,one)),
verify(exor(Z1,d(key(host(C),host(B)),Z2)),
   header(user(A,H),user(D,honest),
     long,two)):
Then:
host_block1 :=
  exor(IV,d(key(host(C),host(B)),Z1)),
host_block2 :=
  exor(Z1,d(key(host(C),host(B)),Z2)),
host_block3 :=
  exor(Z2,d(key(host(C),host(B)),Z3)),
host_block4 :=
  exor(Z3,d(key(host(C),host(B)),Z4)):
event(host(C),
  [host(B),user(A,H),user(D,honest)],
  host_goodguylongdecrypt,
  [iv(ts(host(C),M,N))],M).

Subroutine
  host_honestlongencrypt(host(C),M,N):
host_first := e(key(host(C),host(B)),
  exor(iv(ts(host(C),M,N)),
    header(user(A,honest),user(D,H),
      long,one))),
host_second := e(key(host(C),host(B)),
  exor({host_first},
    header(user(A,honest),user(D,H),
      long,two))),
host_third :=  e(key(host(C),host(B)),
  exor({host_second},
    mess(user(A,honest),user(D,H),
      ts(host(C),M,N),one))),
host_fourth := e(key(host(C),host(B)),
  exor({host_third},
    mess(user(A,honest),user(D,H),
      ts(host(C),M,N),two))):
send msg(host(C),host(D),
  [spi(host(C),host(B)),
    iv(ts(host(C),M,N)),
    {host_first},{host_second},
    {host_third},{host_fourth}],N):
event(host(C),[host(D)],
  host_honestlongencrypt,
  [iv(ts(host(C),M,N))],M).

Subroutine
  host_honestshortencrypt(host(C),M,N) :
send msg(host(C),host(B),
  [spi(host(C),host(B)),
    iv(ts(host(C),M,N)),
    e(key(host(C),host(B)),
      exor(iv(ts(host(C),M,N)),
        header(user(A,honest),user(D,H),
          short,one))),
    e(key(host(C),host(B)),
      exor(e(key(host(C),host(B)),
        exor(iv(ts(host(C),M,N)),
        header(user(A,honest),user(D,H),
          short,one))),
      mess(user(A,honest),user(D,H),
        ts(host(C),M,N),one))),
    e(key(host(C),host(B)),
      exor(e(key(host(C),host(B)),
exor(e(key(host(C),host(B)),
  exor(iv(ts(host(C),M,N)),
  header(user(A,honest),user(D,H),
           short,one))),
mess(user(A,honest),user(D,H),
          ts(host(C),M,N),one))),
      mess(user(A,honest),user(D,H),
        ts(host(C),M,N),two))),
    e(key(host(C),host(B)),
      exor(e(key(host(C),host(B)),
        exor(e(key(host(C),host(B)),
          exor(e(key(host(C),host(B)),
            exor(iv(ts(host(C),M,N)),
            header(user(A,honest),user(D,H),
              short,one))),
          mess(user(A,honest),user(D,H),
            ts(host(C),M,N),one))),
        mess(user(A,honest),user(D,H),
          ts(host(C),M,N),two))),
```

```
      mess(user(A,honest),user(D,H),
        ts(host(C),M,N),three)))],M):
event(host(C),
  [host(B),user(A,honest),user(D,H)],
  host_honestshortencrypt,
  [iv(ts(host(C),M,N))],M).

Subroutine
  host_badguylongencrypt(host(C),M,N) :
If:
rcv msg(user(A,dishonest),host(C),
  [user(D,H),Z1,Z2],M):
Then:
send msg(host(C),host(B),
  [spi(host(C),host(B)),
    iv(ts(host(C),M,N)),
    e(key(host(C),host(B)),
      exor(iv(ts(host(C),M,N)),
      header(user(A,dishonest),user(D,H),
        long,one))),
    e(key(host(C),host(B)),
      exor(e(key(host(C),host(B)),
        exor(iv(ts(host(C),M,N)),
        header(user(A,dishonest),user(D,H),
          long,one))),
      header(user(A,dishonest),user(D,H),
        long,two))),
    e(key(host(C),host(B)),
      exor(e(key(host(C),host(B)),
        exor(e(key(host(C),host(B)),
          exor(iv(ts(host(C),M,N)),
          header(user(A,dishonest),user(D,H),
            long,one))),
        header(user(A,dishonest),user(D,H),
          long,two))),
      Z1)),
    e(key(host(C),host(B)),
      exor(e(key(host(C),host(B)),
        exor(e(key(host(C),host(B)),
          exor(e(key(host(C),host(B)),
            exor(iv(ts(host(C),M,N)),
            header(user(A,dishonest),user(D,H),
              long,one))),
          header(user(A,dishonest),user(D,H),
            long,two))),
        Z1)),
      Z2))],M):
event(host(C),
  [host(B),user(A,dishonest),user(D,H)],
   host_badguylongencrypt,
  [iv(ts(host(C),M,N))],M).



Subroutine
  host_badguyshortencrypt(host(C),M,N) :
If:
rcv msg(user(A,dishonest),host(C),
  [user(D,H),Z1,Z2,Z3],M):
Then:
send msg(host(C),host(B),
  [spi(host(C),host(B)),
    iv(ts(host(C),M,N)),
    e(key(host(C),host(B)),
      exor(iv(ts(host(C),M,N)),
      header(user(A,dishonest),user(D,H),
        short,one))),
    e(key(host(C),host(B)),
      exor(e(key(host(C),host(B)),
        exor(iv(ts(host(C),M,N)),
        header(user(A,dishonest),user(D,H),
          short,one))),
      Z1)),
    e(key(host(C),host(B)),
      exor(e(key(host(C),host(B)),
        exor(e(key(host(C),host(B)),
          exor(iv(ts(host(C),M,N)),
          header(user(A,dishonest),user(D,H),
            short,one))),
        Z1)),
      Z2)),
    e(key(host(C),host(B)),
      exor(e(key(host(C),host(B)),
        exor(e(key(host(C),host(B)),
          exor(e(key(host(C),host(B)),
            exor(iv(ts(host(C),M,N)),
            header(user(A,dishonest),user(D,H),
              short,one))),
          Z1)),
        Z2)),
      Z3))],M):
event(host(C),
  [host(B),user(A,dishonest),user(D,H)],
  host_badguyshortencrypt,
  [iv(ts(host(C),M,N))],M).

Subroutine
  host_badguylongdecrypt(host(B),M,N) :
If:
rcv msg(user(A,H),host(B),
  [spi(host(C),host(B)),IV,Z1,Z2,Z3,Z4],M):
verify(exor(IV,d(key(host(C),host(B)),Z1)),
  header(user(A,H),user(D,dishonest),
    long,one)),
verify(exor(Z1,d(key(host(C),host(B)),Z2)),
  header(user(A,H),user(D,dishonest),
    long,two)):
Then:
send msg(host(B),user(A,H),
  [exor(IV,d(key(host(C),host(B)),Z1)),
    exor(Z1,d(key(host(C),host(B)),Z2)),
    exor(Z2,d(key(host(C),host(B)),Z3)),
    exor(Z3,d(key(host(C),host(B)),Z4))],M):
event(host(C),
  [host(B),user(A,H),user(D,dishonest)],
  host_badguylongdecrypt,
```

```
  [iv(ts(host(C),M,N))],M).

Subroutine
  host_badguyshortdecrypt(host(B),M,N) :
If:
rcv msg(user(A,H),host(B),
  [spi(host(C),host(B)),IV,Z1,Z2,Z3,Z4],M):
verify(exor(IV,d(key(host(C),host(B)),Z1)),
  header(user(A,H),user(D,dishonest),
    short,one)):
Then:
send msg(host(B),user(A,H),
  [exor(IV,d(key(host(C),host(B)),Z1)),
    exor(Z1,d(key(host(C),host(B)),Z2)),
    exor(Z2,d(key(host(C),host(B)),Z3)),
    exor(Z3,d(key(host(C),host(B)),Z4))],N):
event(host(C),
  [host(B),user(A,H),user(D,dishonest)],
  host_badguyshortdecrypt,
  [iv(ts(host(C),M,N))],M).
```

Appendix B: Queries Presented to the Analyzer

Chosen Ciphertext

```
What words is the intruder looking for?
|: d(key(U,V),notsent(P))
|:
What state variable values is the intruder
  looking for?
Give the principal name
|:
List the sequence of events that you want
  to have occurred.
|:
What conditions to you want to put on
  all of these?
|:
List the sequences of events that you dont
  want to have occurred.
Enter a list
|:
```

Chosen Plaintext

```
What words is the intruder looking for?
|:e(key(U,V),notsent(P))
|:
What state variable values is the intruder
  looking for?
Give the principal name
|:
List the sequence of events that you want
  to have occurred.
|:
What conditions to you want to put on
  all of these?
|:
List the sequences of events that you dont
  want to have occurred.
Enter a list
|:
```

Bellovin's Integrity Attack

```
What words is the intruder looking for?
|:
What state variable values is the intruder
  looking for?
Give the principal name
|: host(A)
Give the round number:
|: B
Give the state variables:
|: host_block1 = header(user(D,honest),E,F,G)
|: host_block2 = notsent(P)
|:
Give the principal name
|:
List the sequence of events that you want
  to have occurred.
|:
What conditions to you want to put on
  all of these?
|:
List the sequences of events that you dont
  want to have occurred.
Enter a list
|:
```

Bellovin's Secrecy Attack

```
What words is the intruder looking for?
|: mess(user(A, honest), user(B, honest),
  ts(host(C), D, E), one)
|:
What state variable values is the intruder
  looking for?
Give the principal name
|:
List the sequence of events that you want
  to have occurred.
|:
What conditions to you want to put on
  all of these?
|:
List the sequences of events that you dont
  want to have occurred.
Enter a list
|:
```

Stuart Stubblebine received his BS degree in computer science and mathematics from Vanderbilt University in 1983, his MS degree from University of Arizona in Electrical Engineering in 1988, and his Ph.D. degree in Electrical Engineering from University of Maryland in 1992. After finishing his Ph.D., Dr. Stubblebine was a computer scientist at the University of Southern California's Information Sciences Institute and a research assistant professor in the Department of Computer Science. He remains as an adjunct professor. In 1994 he joined ATT Bell Labs in Murray Hill, NJ and continued with ATT Labs - Research after the ATT breakup in 1996. In 1999, Dr. Stubblebine joined CertCo serving as a Vice President.

Dr. Stubblebine has been active with numerous conferences and journals in security and software engineering. His research interests are rooted in the design, analysis, and formal verification of cryptographic protocols. His recent accomplishments have been in the areas of electronic commerce and privacy, authentication and authorization, trusted third party services, recent-secure revocation in distributed systems, and secure software engineering techniques. Dr. Stubblebine is a member of IEEE and ACM.

Catherine Meadows is head of the Formal Methods Section in the Center for High Assurance Computer Systems at the Naval Research Laboratory She has published more than 50 papers on formal methods, cryptography, and security. Her research interests include the application of formal methods to computer security, in particular to the evaluation of cryptographic protocols and distributed systems. She has also been involved in the organization of numerous conferences on security and reliability, having been program chair of the Computer Security Foundations Workshop, the IEEE Symposium on Security and Privacy, the IEEE Symposium on High Assurance System Engineering, and the Sixth IFIP Working Conference on Dependable Computing for Critical Applications as well as many others. She is also a founding member of IFIP Working Group 1.7 on Foundations of Security Analysis and Design. Prior to coming to NRL, she was an assistant professor of mathematics at Texas A&M University from 1981-1985. Dr. Meadows received a B.A. in mathematics from the University of Chicago in 1975 and a Ph.D. in mathematics from the University of Illinois in 1981.