

# Proof Techniques for Cryptographic Protocols

Kevin J. Compton<sup>1</sup> and Scott Dexter<sup>2</sup>

<sup>1</sup> BRICS, University of Aarhus, DK - 8000 Aarhus C, Denmark and  
EECS Dept., University of Michigan, Ann Arbor, MI 48109-2122, USA  
kjc@umich.edu

<sup>2</sup> Department Of Computer and Information Science, Brooklyn College, City  
University of New York, Brooklyn, New York 11210 USA  
sdexter@sci.brooklyn.cuny.edu

**Abstract.** We give a general introduction to cryptographic protocols and the kinds of attacks to which they are susceptible. We then present a framework based on linear logic programming for analyzing authentication protocols and show how various notions of attack are expressed in this framework.

## 1 Introduction

Cryptographic protocols provide the framework for secure communication in email programs, browsers, remote login facilities, electronic commerce, and many other applications. Not surprisingly, many researchers have worked on providing a rigorous mathematical framework for reasoning about cryptographic protocols. Central to their efforts is the notion of an *attack* on a cryptographic protocol. We give general introduction to cryptographic protocols and the kinds of attacks to which they are susceptible. We illustrate these ideas with four well known protocols. We then present a straightforward method for formalizing protocols within a logic programming framework and argue that linear logic provides a foundation for easily expressing various kinds of attacks. Attacks are easily discovered (or security proved, if there are no attacks) by means of a resolution theorem prover that combines ideas from linear logic programming and equational theorem proving.

## 2 Related Work

Researchers have employed tools from a variety of areas, including logic, algebra, and complexity theory, to analyze cryptographic protocols.

Logics of belief have played an important role, first with the development of BAN logic [11], and then its successors (e.g. [18, 43, 41]). These logics reason about the evolution of belief in the agents participating in authentication and key exchange protocols. Criticisms (e.g. [28, 39, 40]) of these logics center on an informal “idealization” step in which the protocol is modified, the reliance on belief, and the semantics of the logic.

The idea of using algebraic specifications for security protocols is due to Dolev and Yao [16]. In their model, there is an intruder from whose perspective the protocol is a machine for producing words in some language. Symbols in these words correspond to encryption and decryption operations.

Meadows [30] based her work (the “Analyzer”) on that of Dolev and Yao. She noted two limitations of Dolev and Yao’s work: first, that their analysis is limited to a restrictive class of protocols; and second, that adversaries may do more than just discover secret information; many protocols are “broken” when the intruder convinces an agent that information in the intruder’s possession has a certain function, e.g. when the intruder successfully introduces a bogus session key. This method is an extremely powerful tool and has been used to find flaws in a variety of protocols, but it is not designed for proving positive results about security since it does not do a complete search of the search space. We remark that even though the search engine is logic programming based, it is quite different from the logic programming search presented in this paper.

Woo and Lam [45] present an algebraic system in which the specification is given in terms of the protocol each agent follows (rather than in terms of an intruder’s state). Each protocol is a sequence of actions that the agent takes. The authors assert that their model is sufficiently formal to address precisely issues of soundness and completeness, although this is deferred as a question about specific analysis techniques. It is unclear what proofs in this system actually entail, as the correctness assertions need to be proved for all possible executions.

Bolignano [6] has developed a method that has been described [29] as occupying an intermediate position between the techniques of Woo & Lam and Meadows. In this framework, the intruder’s state is represented by the words it can derive, while the agents’ states are simply denoted by program counters. While Bolignano states that his goal is simple and concise proofs (rather than automatability, for example), the proof process becomes unwieldy and must be customized for each protocol. In a later paper [7] some of these problems are rectified.

The process algebra CSP [20] has served as the foundation for a number of investigations of modeling protocols as communicating processes. In [24], the public key variant of the Needham-Schroeder protocol is modeled as a set of CSP processes; this model is tested using FDR, a model checker for CSP. Ryan and Zakiuddin [38] present a summary of methods of formalizing security properties in CSP notation. Bryans and Schneider [9] present a CSP-based analysis of a recursive protocol presented in [10] (and also analyzed in [36]). While all of these analyses benefit from the wide usage CSP enjoys it is also clear that the state space limitations incurred by model-checking sharply affect the quality of the results, although research in this area is ongoing.

Bellare and Rogaway [4, 3, 5] present a complexity-theoretic model of cryptographic protocols that does away entirely with issues of agents’ knowledge, belief, and trust and instead places the intruder at the center. In this model, the intruder is the means by which the players communicate; all messages are sent to the intruder, who may intercept and alter them, or not, as she pleases. While

the framing of this model is very rigorous and produces provably secure protocols, the proofs themselves and the techniques used within them are not easy to automate. Moreover, this technique is designed solely for proving *security*; there is no mechanism for uncovering protocol flaws.

The logic-based approach has proven to be particularly well-suited for automation. Millen’s Interrogator [22] is a Prolog-based search engine that searches for instantiations of a predicate signifying the intruder’s knowledge of specified data via some sequence of messages to an insecure goal state. While the Interrogator has successfully found protocol flaws fairly quickly, search time may vary significantly depending on the precise format of the protocol specification and the amount of information provided about the insecure goal states. Additionally, search heuristics may prevent some flaws from being noticed.

Paulson [34, 35] has developed a model that is tailored for use with Isabelle, an ML-based theorem prover. In his model an intruder can read all messages and send new messages; it can also act as an “honest” agent using its own long-term key. Protocols are modeled by defining the set of possible utterances (both by the honest agents and, possibly, by the intruder), then inductively defining the set of possible protocol traces. Protocol specifications are sets of rules for extending a protocol trace, including a rule describing what the intruder may send. Protocol analysis then consists of proving properties (regularity, uniqueness, secrecy, etc.) of all possible traces.

Marrero *et al.* [29] produced a *model checker* for security protocols. Like Paulson’s work, this approach involves reasoning about all possible protocol traces, but rather than reason about the logical properties of these traces, the model checker performs a state space search and tests the properties of each trace. Security is defined in terms of *secrecy* and *correspondence* properties – each protocol specification includes a list of terms which the intruder must not learn and a mapping between protocol runs of distinct agents.

In this paper we propose linear logic as a framework for security proofs. Mitchell [31] and Cervasato *et al.* [13] have also started applying linear logic in this area.

Much of the recent activity in protocol analysis has focused on defining and characterizing attacks on cryptographic protocols; see [37, 8, 26, 25].

### 3 Protocols and Attacks

A protocol is a script for the exchange of information between two or more individuals or *principals*. The goal of a cryptographic protocol is to allow principals to send information over an open channel with a guarantee of secrecy and authenticity. *Secrecy* means that an intruder with the ability to intercept, alter, and redirect messages will not have access to information the principals wish to protect. *Authenticity* means that the receiver’s assumption about the originator of a message is correct. There may be other goals as well.

Cryptographic protocols may use either a *private* (or *symmetric*) key cryptosystem or a *public key* cryptosystem.

In a private key cryptosystem, the same key is used both for encryption and decryption.  $\{M\}_K$  will denote the encryption of message  $M$  using private key  $K$ . Thus,  $\{\{M\}_K\}_K = M$ . (In some systems encryption and decryption are different functions, but here we will assume they are the same.) Usually, a trusted server  $S$  shares a private key  $K_A$  (called a *long lived key*) with each principal  $A$ . This simplifies key management, since a system with  $n$  principals requires only  $n$  long lived keys, and the addition of a new principal requires the distribution of just one long lived key. The drawback is *key distribution* between a group of two or more principals must take place before they can communicate. Key distribution is often the goal of a cryptographic protocol.

In a public key cryptosystems, each principal  $A$  has a public key  $K_A^+$  which is generally known and by which anyone can send encrypted messages to  $A$ , and a private key  $K_A^-$  which is known only to  $A$ . Obtaining  $K_A^-$  from  $K_A^+$  should be infeasible. These keys are inverses in the sense that  $\{\{M\}_{K_A^+}\}_{K_A^-} = \{\{M\}_{K_A^-}\}_{K_A^+} = M$ .

### 3.1 The Needham-Schroeder Protocol

The (private key) Needham-Schroeder protocol [32] is the canonical example of a flawed protocol.

- (1)  $A \rightarrow S: A, B, N_i$
- (2)  $S \rightarrow A: \{N_i, B, K_i, \{K_i, A\}_{K_B}\}_{K_A}$
- (3)  $A \rightarrow B: \{K_i, A\}_{K_B}$
- (4)  $B \rightarrow A: \{M_i\}_{K_i}$
- (5)  $A \rightarrow B: \{M_i - 1\}_{K_i}$
- (A)  $A \rightarrow B: \{P_i\}_{K_i}$
- (B)  $B \rightarrow A: \{Q_i\}_{K_i}$

The principals are  $A$ ,  $B$ , and  $S$ .  $A$  wishes to establish a private communication with  $B$  on an open channel, but  $A$  and  $B$  do not share a key. Since this protocol may be run many times and by many different principals, we designate a session number  $i$  which is used as an index for all information particular to the session.

In step (1),  $A$  sends a message to the  $S$  indicating that  $A$  wishes to initiate communication with  $B$ .  $A$  includes a nonce  $N_i$ . A *nonce* is a string of random bits, usually freshly generated, sent a principal with a message in the expectation that the nonce will accompany a response. In step (2),  $S$  responds with a message encrypted under the key  $K_A$ . This message contains  $A$ 's nonce, the new session key  $K_i$ , and a message that  $A$  will forward to  $B$ . In (3),  $A$  forwards this message to  $B$ . This message, encrypted under  $K_B$ , contains the  $K_i$  and indicates that it is to be used to communicate with  $A$ . The next two messages are a *handshake* indicating that the parties have received the new key:  $B$  sends a new nonce  $M_i$  to  $A$ , encrypted under the new key;  $A$  then decrypts the message and replies with  $M_i - 1$  encrypted under the new key. One of the implicit assumptions about the cryptosystem is that only the possessor of  $K_i$  could make such a reply. The principals then proceed with their session. We assume that the session key will

be used at some point by either or both agents to encrypt secret information; we model this information explicitly as  $P_i$  and  $Q_i$  in steps (A) and (B). Strictly speaking, these steps are not part of the protocol and do not appear in most presentations. We include them here and subsequent examples as a convenience.

The Denning-Sacco attack [14] on this protocol assumes that an intruder  $I$  has obtained a message of the form  $\{k, A\}_{K_B}$  and knows  $k$ , perhaps by analyzing an old session. (This is an example of a *replay attack*.)  $I$  masquerades as  $A$  by sending this message to  $B$ , who will interpret this as the third message in the protocol. Since  $I$  knows  $k$ , he can trick  $B$  into completing the “handshake”, and  $B$  will believe he has mutually authenticated with  $A$ .  $B$  then divulges the secret  $Q_i$ . Here is the description of the attack.  $I$  initiates a bogus session  $\alpha$ . We will write  $I_A$  to indicate that  $I$  is masquerading as  $A$ .

$$\begin{aligned} (3\alpha) \quad I_A &\rightarrow B: \{k, A\}_{K_B} \\ (4\alpha) \quad B &\rightarrow I_A: \{M_i\}_k \\ (5\alpha) \quad I_A &\rightarrow B: \{M_i - 1\}_k \\ (\text{A}\alpha) \quad B &\rightarrow I_A: \{Q_i\}_k \end{aligned}$$

$I$  then knows  $Q_i$ .

### 3.2 The Public Key Needham-Schroeder Protocol

In this protocol principals  $A$  and  $B$  use a public key cryptosystem to share a secret which could be used, for example, as a shared private key. The secret is a pair of nonces  $N_i$  and  $M_i$  generated by  $A$  and  $B$ , respectively.

$$\begin{aligned} (1) \quad A &\rightarrow B: \{A, N_i\}_{K_B^+} \\ (2) \quad B &\rightarrow A: \{N_i, M_i\}_{K_A^+} \\ (3) \quad A &\rightarrow B: \{M_i\}_{K_B^+} \\ (\text{A}) \quad A &\rightarrow B: \{P_i\}_{[N_i, M_i]} \\ (\text{B}) \quad B &\rightarrow A: \{Q_i\}_{[N_i, M_i]} \end{aligned}$$

Lowe [24] found the following *man-in-the-middle* attack.  $A$ , unaware that  $I$  is malicious, begins a protocol session with  $I$ .

$$(1\alpha) \quad A \rightarrow I: \{A, n\}_{K_I^+}$$

$I$  decrypts this message with his private key, encrypts it with the public key of  $B$ , and begins a masquerade as  $A$ .

$$\begin{aligned} (1\beta) \quad I_A &\rightarrow B: \{A, n\}_{K_B^+} \\ (2\beta) \quad B &\rightarrow I_A: \{n, M_i\}_{K_A^+} \end{aligned}$$

$I$  now returns to the original session with  $A$ .

$$\begin{aligned} (2\alpha) \quad I &\rightarrow A: \{n, M_i\}_{K_A^+} \\ (3\alpha) \quad A &\rightarrow I: \{M_i\}_{K_I^+} \end{aligned}$$

$I$  then returns to the protocol session with  $B$ , completing the masquerade.

$$\begin{aligned} (3\beta) \quad I_A &\rightarrow B: \{M_i\}_{K_B^+} \\ (\text{B}\beta) \quad B &\rightarrow I_A: \{Q_i\}_{[n, M_i]} \end{aligned}$$

### 3.3 The Wide Mouthed Frog Protocol

This intriguingly named protocol first appeared in [11]. It incorporates a novel combination of techniques to achieve its small size. For example,  $A$  rather than  $S$  generates the session key  $K_i$ . Freshness is guaranteed by use of *timestamps*  $T_1$  and  $T_2$ . Timestamps are common in cryptographic protocols. Usually the duration a timestamped message is valid is several hours, so clock synchronization is not a problem.

- (1)  $A \rightarrow S: A, \{T_1, B, K_i\}_{K_A}$
- (2)  $S \rightarrow B: \{T_2, B, K_i\}_{K_B}$
- (A)  $A \rightarrow B: \{P_i\}_{K_i}$
- (B)  $B \rightarrow A: \{Q_i\}_{K_i}$

Lowe [24] proposed an attack in which  $I$  eavesdrops on a session above and later sends another copy of message 2.

- (2 $\alpha$ )  $I_S \rightarrow B: \{T_2, B, K_i\}_{K_B}$

Is this really an attack? In contrast to the previous attacks, messages  $P_i$  and  $Q_i$  have not been compromised, and there is no indication that  $I$  could send a bogus message  $R_i$  masquerading as one of the principals. However, in session  $\alpha$ ,  $B$  believes he has successfully completed a (second) session of the protocol, but he has not.

Lowe presents several attacks using essentially the same idea. These attacks can usually be thwarted by adding a handshake.

### 3.4 The Bellare-Rogaway 3PKD Protocol

This protocol [5] uses a cryptographic hash to reduce the size. A principal  $A$  may authenticate a message  $M$  to  $B$  by sending an encryption  $\{M\}_K$ , where  $K$  is a key shared between  $A$  and  $B$ . However, it may not be important that  $\{M\}_K$  be decipherable; in some situations  $M$  may have been sent as clear text and  $\{M\}_K$  is simply a *message authentication code*, a kind of signature. In these cases computing a *keyed hash function*  $\langle M \rangle_K$  may be much easier to implement and much faster. Given a key  $K$  and message  $M$ , it should be easy to compute the hash value, but without  $K$  it should be infeasible.

- (1)  $A \rightarrow B: A, N_i$
- (2)  $B \rightarrow S: A, B, N_i, M_i$
- (3)  $S \rightarrow A: \{K_i\}_{K_A}, \langle A, B, N_i, \{K_i\}_{K_A} \rangle_{K_A}$
- (4)  $S \rightarrow B: \{K_i\}_{K_B}, \langle A, B, N_i, \{K_i\}_{K_B} \rangle_{K_B}$
- (A)  $S \rightarrow A: \{P_i\}_{K_i}$
- (B)  $B \rightarrow A: \{Q_i\}_{K_i}$

Bellare and Rogaway prove that this protocol secure using a “partner function” definition of security. Notice, though, that the protocol is susceptible to the same kind of attack as we saw for the Wide Mouthed Frog Protocol.

- (1 $\alpha$ )  $I_A \rightarrow B: A, N_i$
- (2 $\alpha$ )  $B \rightarrow S: A, B, N_i, M_i$
- (3 $\alpha$ )  $S \rightarrow I_A: \{K_i\}_{K_A}, \langle A, B, N_i, \{K_i\}_{K_A} \rangle_{K_A}$
- (4 $\alpha$ )  $S \rightarrow B: \{K_i\}_{K_B}, \langle A, B, N_i, \{K_i\}_{K_B} \rangle_{K_B}$

$B$  now believes that he has successfully completed the protocol with  $A$ , but he has not.

## 4 Formalizing Protocol Analysis

In this section we develop a formalism for describing protocols and proving them correct. The goal, as we have seen in the examples of the previous section, is to give a definition of attack that is simple, easy to verify, and takes into account the attacks proposed in the literature.

### 4.1 The Vocabulary of a Protocol Session

From the examples above we see that there are various subtypes partitioning the type **Messages**.

- **Names** of principals ( $A, B, \text{etc.}$ ) are public and *session independent* (*i.e.*, not subscripted).
- **Keys** may either be private and session independent or private and session dependent.
- **Secrets** include nonces and messages such as  $P_i$  and  $Q_i$ . Secrets are session dependent and private.
- **Timestamps** are used to restrict the availability of information to a particular time interval.

We build complex messages from simpler messages using various operations.

- The *encryption operation*  $E(K, M) = \{M\}_K$ .
- The long lived key operation  $K(A) = K_A$ . For public key protocols there are two operations  $K^+(A)$  and  $K^-(A)$ .
- The *handshake operation*  $F(M)$ .
- The *tupling operation* applied to a sequence of messages  $M_1, \dots, M_k$  (with  $k \geq 0$ ), produces a list  $[M_1, \dots, M_k]$ .
- The *keyed hash function*  $H(K, M) = \langle M \rangle_K$ .
- The bitwise exclusive-or function  $X(N, M)$ , which was not used in any of the protocols discussed here, but does appear often in protocols.

We may specify a vocabulary of constant and function symbols for a particular session  $i$  of a protocol. In the Needham-Schroeder protocol the constant symbols are  $A, B, S$  of type **Names**;  $K_A, K_B, K_i$  of type **Keys**; and  $N_i, M_i, P_i$  and  $Q_i$  of subtype **Secrets**. The unary function symbols are  $K$  of type **Names**  $\rightarrow$  **Keys** and  $F$  of type **Secrets**  $\rightarrow$  **Secrets**. The only binary function symbol is  $E$  of type **Keys**  $\times$  **Secrets**  $\rightarrow$  **Secrets**. The tupling function is of type **Messages**\*  $\rightarrow$  **Secrets**.

## 4.2 The Equational Theory of a Cryptosystem

The equational theory  $\Gamma$  for private key cryptosystems contains the axiom

$$(\Gamma 1) \ E(k, E(k, x)) = x$$

Free variables in axioms are assumed to be universally quantified. When we model a public key cryptosystem or use the exclusive-or operation, other axioms are needed.

Since  $\Gamma$  is an equational theory, it has an *initial structure*  $\mathbf{A}_\Gamma$  (see [27]); *i.e.*, there is a unique structure  $\mathbf{A}_\Gamma \models \Gamma$  such that for every  $\mathbf{B} \models \Gamma$ , there is a unique homomorphism from  $\mathbf{A}_\Gamma$  to  $\mathbf{B}$ . Furthermore,  $\mathbf{A}_\Gamma$  is a *term structure*: every element is interpreted by some closed term. Let  $\mathbf{A}$  be the set of all closed terms over the vocabulary (sometimes this is called the *Herbrand universe*) and  $\sim$  be the equivalence relation holds between closed terms  $t_1$  and  $t_2$  if and only if  $\Gamma \vdash t_1 = t_2$ . Then  $\mathbf{A}_\Gamma$  is the quotient structure  $\mathbf{A} / \sim$ . We will refer to  $\mathbf{A}_\Gamma$  as the *message universe* of the protocol. Our methods assume *strong encryption*: distinct elements of the message universe represent items of information not easily computable from one another. This is a difficult property to verify, but one made implicitly not only in security proofs, but even in the design of protocols. For example, the Needham-Schroeder handshake assumes that  $\{M\}_K$  and  $\{M - 1\}_K$ , represented as distinct elements  $E(K, M)$  and  $E(K, F(M))$  in the message universe, are not easily computed from each other.

## 4.3 The Horn Theory of the Passive Intruder

Intruder-based methods for verifying protocol security reason about the information that an intruder can gain. An intruder can gain information by passive means, such as eavesdropping and using public information, or by active means such as manipulating the flow of information, impersonating other agents, and using old information from other sessions. The passive means by which an intruder can gain information can be written as a universal Horn theory  $\Delta$  describing a unary relation  $I$  representing the information accessible to the intruder. In anticipation of the theorem proving techniques we will be using, we express the universal Horn sentences in a logic programming syntax. (Lower case letters are variables and are universally quantified.)

$$\begin{aligned} (\Delta 1) \ & I(E(k, x)) \leftarrow I(k), I(x) \\ (\Delta 2) \ & I([x_1, \dots, x_k]) \leftarrow I(x_1), \dots, I(x_k) \text{ for every } k \geq 0. \\ (\Delta 3) \ & I(x_i) \leftarrow I([x_1, \dots, x_k]) \text{ for every } k \geq i \geq 1. \\ (\Delta 4) \ & I(x) \leftarrow I(K(x)) \\ (\Delta 5) \ & I(x) \leftarrow I(F(x)) \\ (\Delta 6) \ & I(F(x)) \leftarrow I(x) \end{aligned}$$

In addition, under the label  $(\Delta 0)$  we include trivial axioms  $I(A)$ ,  $I(B)$ , and  $I(S)$ , since names are public. Notice that in the private key framework, the Horn sentence  $I(x) \leftarrow I(x), I(E(k, x))$  follows from  $(\Gamma 1)$  and  $(\Delta 1)$ . If the keyed hash function  $H$  is used in the protocol, it is also necessary to include the axiom

$$(\Delta 7) \ I(H(k, x)) \leftarrow I(k), I(x)$$

$\Gamma \cup \Delta$  can be regarded as a logic program (with equations) whose least fixpoint semantics (see [23]) on the message universe defines a relation  $I$  representing the intruder's passive knowledge. Inductive definability is a recurrent theme in protocol security proofs. For example, logics of knowledge and belief operate on the principle that a principal's knowledge is inductively defined, and Paulson [2, 34–36] bases his method on inductively defined relations.

#### 4.4 Linear Logic and the Active Intruder

We now have a fairly clear picture of the passive means by which the intruder can obtain information. The active means involve manipulating the network and sending false messages. For example, in steps (3) and (4) of the Needham-Schroeder protocol,  $B$  receives a message  $\{K_i, A\}_{K_B}$  and sends a message  $\{M_i\}_{K_i}$ . Now  $B$  really has no idea who sent this message. If the intruder sends a message of the form  $\{k, A\}_{K_B}$ ,  $B$  will reply  $\{M_i\}_k$ . Later, in steps (5) and (B), if the intruder sends a message of the form  $\{M_i - 1\}_k$ ,  $B$  will reply  $\{Q_i\}_k$ . It seems that we need two more Horn sentences.

$$\begin{aligned} \text{(B1)} \quad & I(E(k, M_i)) \leftarrow I(E(K(B), [k, A])) \\ \text{(B2)} \quad & I(E(k, Q_i)) \leftarrow I(E(K(B), [k, A])), I(E(k, F(M_i))) \end{aligned}$$

Notice that the second sentence says that the intruder cannot trick  $B$  into responding unless it can supply both of the messages  $B$  expects to receive in previous steps of the protocol.

There are several problems here. We do not know that  $k$  in the first sentence is the same  $k$  as in the second sentence. The first sentence may be used several times in a proof. Thus, from  $E(K(B), [k, A])$  the intruder may learn  $E(k, M_i)$  and from  $E(K(B), [k', A])$  the intruder may learn  $E(k', M_i)$ . This is not correct. The second time the rule is used,  $B$  generates a new nonce  $M_j$  and replies  $E(k', M_j)$ . It is possible to express this distinction using first-order logic, but the analysis becomes much more complicated. Instead, we observe that from the intruder's viewpoint, agents' responses are limited resources; the appropriate logic to describe limited resources is linear logic [17]. Mitchell *et al.* [31] observed that linear logic is a useful tool to reason about cryptographic protocols. We will take a familiarity with the basics of linear logic for the remainder of the paper.

To see how linear logic pertains to security, take the conjunction of the Horn formulas describing how the intruder can derive information from a given principal, preface with  $!$  and then universally quantify. In the traditional linear logic syntax we would obtain the following from (B1) and (B2) above.

$$\begin{aligned} \forall k! \quad & (I(E(K(B), [k, A])) \multimap I(E(k, M_i)) \otimes \\ & (I(E(K(B), [k, A])) \otimes I(E(k, F(M_i))) \multimap I(E(k, Q_i)))) \end{aligned}$$

Reasoning about this linear logic sentence more accurately reflects how the intruder obtains information. Recall how the left  $\forall$ -introduction rule works in

linear logic.

$$\frac{\Sigma, \psi(k \leftarrow t) \vdash \Pi}{\Sigma, \forall k \psi(k) \vdash \Pi}$$

We assume that  $\Sigma$  and  $\Pi$  are multi-sets of linear logic formulas,  $k$  is a variable, and  $t$  is a term. Also,  $\psi(k \leftarrow t)$  is formed by substituting  $t$  for free occurrences of  $k$  in  $\psi$  (subject to the usual condition that variable occurrences in  $t$  must be free wherever  $t$  is substituted). When we work our backward through a linear logic proof, we see that the variable  $k$  in  $\psi(k)$  *can be instantiated only once*. This instantiation determines the information the intruder requires to send a bogus message to  $B$  and receive replies.

Consequently, we should view (B1) and (B2) as as a unit ( $\Delta_B$ ) rather than separate Horn sentences. Such units will be called *definite program templates*. We will give a precise definition in the next section, but first we present the other definite program templates derived from the Needham-Schroeder protocol. Template ( $\Delta_A$ ) consists of the following Horn sentences.

- (A1)  $I([A, B, N_i]) \leftarrow$
- (A2)  $I(y) \leftarrow I(E(K(A), [N_i, B, k, y]))$
- (A3)  $I(E(k, F(m))) \leftarrow I(E(K(A), [N_i, B, k, y])), I(E(k, m))$
- (A4)  $I(E(k, P_i)) \leftarrow I(E(K(A), [N_i, B, k, y])), I(E(k, m))$

Template ( $\Delta_S$ ) consists of the following Horn sentence.

- (S1)  $I(E(K(a), [n, b, k, E(K(b), [k, a])))) \leftarrow I([a, b, n])$

#### 4.5 Linear Logic Programming

We first extend the usual logic programming terminology. These definitions could be somewhat more general, but they will suffice for security proving applications.

A (linear) *definite program clause* is of the form  $\psi \leftarrow \varphi_1, \dots, \varphi_k$ , where  $\varphi_1, \dots, \varphi_k, \psi$  are non-equational atomic formulas. Formula  $\psi$  is the *head* of the clause and formula sequence  $\varphi_1, \dots, \varphi_k$  is the *body* of the clause. A (linear) *definite template* is a set of definite program clauses. A *definite program* is a set of equations, definite program clauses, and definite templates. A (linear) *goal clause* is of the form  $\varphi_1, \dots, \varphi_k$ , where  $\varphi_1, \dots, \varphi_k$  are non-equational atomic formulas.

Given a substitution  $\sigma$  mapping variables to terms we denote by  $E\sigma$  (where  $E$  is either a term, definite clause, or definite template) the result obtained by replacing variables with their images under  $\sigma$ . Substitution  $\sigma$  is a *unifier* for expressions  $E$  and  $F$  if  $E\sigma$  and  $F\sigma$  are syntactically identical;  $\sigma$  is the *most general unifier* (written  $mgu(E, F)$ ) if, in addition, whenever  $\theta$  is a unifier of  $E$  and  $F$ , there is a substitution  $\rho$  such that  $\theta = \sigma\rho$ .

As we have seen, cryptographic protocols can be translated directly into definite programs.

When equations are incorporated into logic programs, different mechanisms may be added to SLD-resolution to obtain a complete theorem proving system.

Many of these use modifications of the *paramodulation* rule (see [33] for a discussion). There are also many logic programming theorem provers for fragments of linear logic [1, 12, 19, 21, 42, 44]. The following resolution theorem prover combines ideas from these two areas and is a complete theorem proving system for the fragment of linear logic used here. It is easier to state as a nondeterministic process which can then be transformed into a backtracking resolution search (as found, for example, in pure Prolog). Without loss of generality we assume that initially the variables in each definite template are unique to that template.

**Given:** Definite program  $\Pi$  and goal clause  $\gamma$   
**Find:** Substitution  $\sigma$  such that  $\Pi \vdash \gamma\sigma$   
Initialize  $\sigma =$  identity substitution  
**Repeat** until  $\gamma$  is empty  
  **Choose** one:  
    Equation  $s = t$  or  $t = s$  from  $\Pi$ :  
      **Choose** subterm  $u$  of  $\gamma$  unifiable with  $t$   
      Let  $\gamma = \delta(x \leftarrow u)$  and  $\rho = mgu(t, u)$   
      Replace  $\gamma$  with  $(\delta(x \leftarrow s))\rho$   
    Definite clause  $\varphi \leftarrow \psi_1, \dots, \psi_k$  from  $\Pi$  or a template:  
      **Choose** subformula  $\vartheta$  of  $\gamma$  unifiable with  $\varphi$   
      Let  $\gamma = \beta \cup \{\vartheta\}$  and  $\rho = mgu(\varphi, \vartheta)$   
      Replace  $\gamma$  with  $(\beta \cup \{\psi_1, \dots, \psi_k\})\rho$   
  Replace each definite template  $\Lambda$  with  $\Lambda\rho$   
  Replace  $\sigma$  with  $\sigma\rho$

The novelty is that the definite templates are modified as the proof proceeds; i.e., the logic program modifies itself. This reflects the linear logic philosophy that a formula is a proof resource and is modified whenever it is used.

## 5 Defining Attack

We conclude the paper by describing how the resolution theorem prover above finds the attacks on protocols in the previous sections.

If we run the resolution theorem prover above on the definite program  $\Pi$  for the Needham-Schroeder with the goal  $I(Q_i)$ , we do not find (as we might have hoped) a proof corresponding to the Denning-Sacco attack. In fact, if we add some obvious heuristics to prune infinite branches in the resolution search process, we discover that  $I(Q_i)$  does not follow from  $\Pi$ : the intruder should not be able to learn  $Q_i$ .

To see what goes wrong, use the attack (steps (3 $\alpha$ ), (4 $\alpha$ ), (5 $\alpha$ ) and (A $\alpha$ ) in section 3.1) to guide the nondeterministic choices made by the theorem prover. The result is pictured as a tree in Figure 1. Each node is labeled by a formula. For each formula  $\vartheta$  on an internal node, a definite clause  $\varphi \leftarrow \psi_1, \dots, \psi_k$  is chosen,  $\rho = mgu(\varphi, \vartheta)$  is computed, and the children of the node are labeled  $\psi_1\rho, \dots, \psi_k\rho$ . The problem is apparent: this is not a complete proof. The leaves

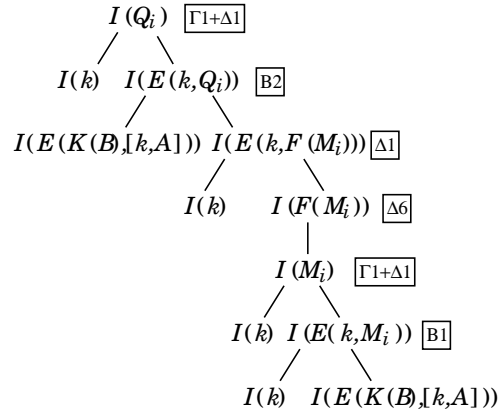


Fig. 1. Proof giving the Denning-Sacco Attack

the tree should have empty labels, but here they are labeled either by  $I(k)$  or  $I(E(K(B), [k, A]))$ . This is to be expected since we assume that  $I$  knows  $k$  and  $\{k, A\}_{K_B}$ . In fact, Figure 1 shows that  $\Pi, I(k), I(E(K(B), [k, A])) \vdash I(Q_i)$ . This suggests that the key to finding replay attacks is to use some condition other than goal clause emptiness to terminate the resolution process. Examining this resolution proof and the ones that result from the Public Key Needham-Schroeder Protocol and other protocols, we see a pattern. Formulas at the leaves of the tree contain no session dependent information: there are no occurrences of the session number index  $i$ . Replay attacks are discovered by changing the loop termination condition to session independence of  $\gamma$ . Since the intruder can mount uninteresting attacks by stealing long lived keys, our definition of *session independence* excludes formulas of the form  $I(K(A))$  as well formulas containing session subscripted information.

To formalize protocols with timestamps we introduce a predicate  $C(T)$  indicating that the timestamp  $T$  is current. In this case we say that a goal clause is *time independent* if, whenever it contains a formula  $C(T)$ , it contains no other occurrences of  $T$  except perhaps the formula  $I(T)$  (since  $I$  can always produce a current timestamp). The termination condition in the resolution theorem prover now requires both session and time independence of the goal.

We now have a general framework for defining and finding attacks. As we have seen, we find violations of *secrecy* by setting the goal to be either  $I(P_i)$  or  $I(Q_i)$ , where  $P_i$  and  $Q_i$  are messages sent by  $A$  and  $B$  with the distributed key.

Violations of *authenticity* are more subtle. Consider the Needham-Schroeder definite program. To show that  $I$  can originate a message which  $B$  thinks came from  $A$ , we add a new program clause  $I(R_i)$  and set  $\gamma$  to be  $I(E(k, Q_i)), I(E(k, R_i))$ . The first formula ensures that  $B$  has successfully completed the protocol and accepted  $k$ . The second formula says that  $I$  can

encrypt a known message with  $k$ . In this kind of situation we say that *weak key authenticity* has been violated.

Authenticity might be violated in another way.  $I$  might send  $B$  a message encrypted with the distributed key, but not know the message. In that case, the goal should be  $I(E(k, Q_i)), I(E(k, m))$ . This goal is satisfied by any protocol that succeeds in distributing a session key.  $I$  simply transmits messages sent out by the principals to the proper receivers and at the conclusion  $m = Q_i$ . However, if there is another value that satisfies the goal, the theorem prover will find it. In this kind of situation we say that *strong key authenticity* has been violated.

Observe that the definite templates have been modified during the resolution process and that when the process terminates, the templates describe the information each principal has received and sent. For example, at the end of the Denning-Sacco attack, substitutions will have been made in the variables of template  $\Lambda_B$  to match the goal variables. (Recall that initially the variables in a template are unique to that template.) However, the variables in  $\Lambda_A$  are unchanged, indicating that the attack was played without the participation of  $A$ .

We can use the same idea to uncover Lowe's attack on the Wide Mouthed Frog Protocol. Set  $\gamma$  to be  $I(E(k, Q_i))$ . Again, since the protocol succeeds in distributing a key, the goal will be satisfied. However, for some proofs, the variables in  $\Lambda_A$  are unchanged, indicating that  $A$  did not initiate a protocol session, even though  $B$  believes she did. The same kind of analysis uncovers the attack we described on the Bellare-Rogaway 3PKD protocol.

We have implemented a limited version of the resolution theorem prover described here and tested secrecy and weak key authenticity for many protocols [15]. This system provides a straightforward conversion from conventional informal protocol specifications to formal specifications and allows us to perform fast automated analysis. As we have seen, another advantage is that it should be easily adaptable to find other kinds protocol attacks. The system uses heuristics to prune infinite resolution searches and is therefore capable of proving security as well as finding attacks. Since the heuristics pertain only to the program clauses in  $\Gamma$  and  $\Delta$ , they do not have to be tailored to a particular protocol. The system has taken less than one second of CPU time on every protocol we have given it, and has always produced an answer – either by displaying an attack or certifying security of the protocol.

## References

1. J.-M. Andreoli. Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation*, 2(3), 1992.
2. G. Bella and L. C. Paulson. Using Isabelle to prove properties of the Kerberos authentication system. In *Proceedings of the DIMACS Workshop on Formal Verification of Cryptographic Protocols*, September 1997.
3. M. Bellare and P. Rogaway. Entity authentication and key distribution. In *Advances in Cryptology — Crypto '93 Proceedings*, 1993.
4. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the First ACM Conference on Computer and Communications Security*, 1993.

5. M. Bellare and P. Rogaway. Provably secure session key distribution — the three party case. In *Proceedings of the 27th ACM Symposium on the Theory of Computing*, 1995.
6. D. Bolignano. An approach for the formal verification of cryptographic protocols. In *Proceedings of the Third ACM Conference on Computer and Communications Security*, pages 106–118. ACM Press, 1996.
7. D. Bolignano. Towards a mechanization of cryptographic protocol verification. In *Proceedings of the 9th International Computer-Aided Verification Conference*, June 1997.
8. C. Boyd. Towards extensional goals in authentication protocols. Preprint.
9. J. Bryans and S. Schneider. CSP, PVS, and a recursive authentication protocol. In *Proceedings of the DIMACS Workshop on Formal Verification of Cryptographic Protocols*, September 1997.
10. J. A. Bull and D. J. Otway. The authentication protocol. Technical Report CSM/436-04/03, Defence Research Agency, Malvern, UK, 1997.
11. M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8, February 1990.
12. S. Cerrito. Herbrand methods in sequent calculi: Unification in LL. In K. Apt, editor, *Logic Programming: Proceedings of the Joint International Conference and Symposium on Logic Programming*, pages 607–621. The MIT Press, 1992.
13. I. Cervesato, N. A. Durgin, P. D. Lincoln, J. C. Mitchell, and A. Scedrov. A meta-notation for protocol analysis. Preprint.
14. D. E. Denning and G. M. Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24(8):533–536, 1981.
15. S. D. Dexter. *An Adversary-Centric Logic of Security and Authenticity*. PhD thesis, University of Michigan, 1998.
16. D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, IT-29(2):198–208, March 1983.
17. J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.
18. L. Gong, R. Needham, and R. Yahalom. Reasoning about belief in cryptographic protocols. In *Proceedings of the IEEE Computer Society Symposium on Security and Privacy*, pages 234–248. IEEE, May 1990.
19. J. Harland and D. Pym. A uniform proof-theoretic investigation of linear logic programming. *Journal of Logic and Computation*, 4(2), April 1994. 175–207.
20. C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.
21. J. S. Hodas and D. Miller. Logic programming in a fragment of intuitionistic linear logic. *Journal of Information and Computation*, 110(2):327–365, May 1994.
22. R. Kemmerer, C. Meadows, and J. Millen. Three systems for cryptographic protocol analysis. *Journal of Cryptology*, 7:79–130, 1994.
23. J. W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, second extended edition edition, 1993.
24. G. Lowe. Breaking and fixing the Needham-Schroeder public key protocol using CSP and FDR. In T. Margaria and B. Steffen, editors, *Tools and Algorithms for the Construction and Analysis of Systems Second International Workshop, TACAS '96*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer-Verlag, 1996.
25. G. Lowe. A hierarchy of authentication specification. In *Proceedings of the 10th IEEE Computer Security Foundations Workshop*, June 1997.
26. G. Lowe. Some new attacks upon security protocols. In *Proceedings of the 9th IEEE Computer Security Foundations Workshop*, pages 162–169, 1997.

27. J. A. Makowsky. Why Horn formulas matter in computer science: Initial structures and generic examples. *Journal of Computer and System Sciences*, 34:266–292, 1987.
28. W. Mao and C. Boyd. Development of authentication protocols: Some misconceptions and a new approach. In *Proceedings of the Computer Security Foundations Workshop VII*, pages 178–186. IEEE, 1994.
29. W. Marrero, E. Clarke, and S. Jha. Model checking for security protocols. Technical Report CMU-CS-97-139, School of Computer Science, Carnegie Mellon University, May 1997.
30. C. Meadows. The NRL protocol analyzer: an overview. *The Journal of Logic Programming*, pages 113–131, 1996.
31. J. C. Mitchell. Analysis of security protocols. Slides for a talk at CAV '98, available at <http://www.stanford.edu/~jcm>, July 1998.
32. R. M. Needham and M. D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.
33. P. Padawitz. *Computing in Horn clause theories*. Springer-Verlag, Berlin, 1988.
34. L. C. Paulson. Proving properties of security protocols by induction. Technical Report TR-409, Computer Laboratory, University of Cambridge, 1996.
35. L. C. Paulson. Mechanized proofs for a recursive authentication protocol. In *Proceedings of the 10th IEEE Computer Security Foundations Workshop*, pages 84–95, June 1997.
36. L. C. Paulson. Mechanized proofs of security protocols: Needham-Schroeder with public keys. Unpublished manuscript, January 1997.
37. A. W. Roscoe. Modelling and verifying key exchange protocols using CSP and FDR. In *Proceedings of the Computer Security Foundations Workshop VIII*, volume 8, pages 98–107. IEEE, 1995.
38. P. Ryan and I. Zakiuddin. Modelling and analysis of security protocols. In *Proceedings of the DIMACS Workshop on Formal Verification of Security Protocols*, September 1997.
39. E. Sneekenes. Exploring the BAN approach to protocol analysis. In *Proceedings of the IEEE Computer Society Symposium on Security and Privacy*, pages 171–181, May 1991.
40. P. Syverson. The use of logic in the analysis of cryptographic protocols. In *Proceedings of the 1991 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 156–170. IEEE, May 1991.
41. P. Syverson and P. van Oorschot. On unifying some cryptographic protocol logics. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 14–28, 1994.
42. T. Tammet. Proof strategies in linear logic. *Journal of Automated Reasoning*, 12:273–304, 1994.
43. P. C. van Oorschot. Extending cryptographic logics of belief to key agreement protocols (extended abstract). In *Proceedings of the First ACM Conference on Computer and Communications Security*, pages 232–243, November 1993.
44. M. Winikoff and J. Harland. Some applications of the linear logic programming language lygon. In *Proceedings of the Australasian Computer Science Conference*, pages 262–271, February 1996.
45. T. Y. C. Woo and S. S. Lam. A semantic model for authentication protocols. In *Proceedings of the Symposium on Research in Security and Privacy*, pages 178–194. IEEE, May 1993.