To the memory of my father,

Andrew Yasinsac (1909-1990)

Abstract

Secure communication is largely dependent on effective application of cryptography. While cryptographic methods have been investigated to ensure confidence in the security of the encryption algorithm, many cryptographic schemes are vulnerable to attack because the protocols used to implement communication in a cryptographic environment do not meet their intended goals. Cryptographic protocols, like software, are very difficult to verify. Recent research is aimed at finding methods of verifying cryptographic protocols. Though no method has achieved widespread acceptance and use, the most prevalent class of cryptographic protocol evaluation techniques is based on application of epistemic logical systems to reason about the protocols. In this dissertation we present a methodology for verifying cryptographic protocols based on the classical program verification technique of Weakest Precondition reasoning [DIJK76] and a set of tools to automate application of the method. The methodology solves many problems of existing methods by formalizing the meaning of the messages in a protocol session and by including message sequencing in the protocol definition. The proposed method can also be combined with logical methods to construct a more thorough cryptographic protocol evaluation technique.

Acknowledgements

Before anything else, I must thank my wife Linda for her sacrifice, help, support, encouragement and perseverence during this process. Clearly she is a chief contributor to this dissertation in many ways and without her it never would have happened. My children, Bryan and Cristin, also made significant sacrifices to allow me to continue this work. While we can never regain the family time lost, hopefully we will benefit in the future from the emphasis we have had on cherishing the time we do share.

There are a large number of people that contributed to the process that lead to this dissertation and there is no way to name them all. While I'll limit this acknowledgement to those that have been closest to the work, I have appreciated the encouragement and advice that I have recieved from all my friends and family.

Alf Weaver provided my initial guidance in the department and supported me to the very end. His advice was crutial in getting me on the right track for finding a suitable area of research. Andrew Grimshaw helped me to understand just how significant an undertaking this was, and Jack Davidson showed me that there are no shortcuts just because I may have been unprepared in some areas. Paul Syverson's work motivated my start in this area and his encouragement was key to my staying the course. Alan Batson, Steve Strickland, and Jorg Liebeherr provided valuable comments on the direction of the research and on the final paper.

Special thanks are due to those in my research group for the adjustments they

made to accomodate my personal circumstances. A more understanding group has never existed. Ramesh Peri, Brett Tjaden, Katie Oliver, Chenxi Wang, and most recently, Darrell Kienzle are full contributors to this dissertation. Their insights, comments, ideas, and criticisms did more than help shape the work, they helped build it; as did Sally McKee. Sally is a devoted friend above all else. She doesn't know how to say no to a request from a friend.

My mother, brother Andrew, father-in-law Phil Coker and a host of extended family never doubted the final results. A whole host of friends starting with the Atchisons, Guertlers, Beckmans and Lee McCall encouraged and supported us in many ways over this long haul.

Finally, I cannot over-emphasize my appreciation to Bill Wulf for the patience and understanding he showed and the sacrifices he made to help me through. Aside from the inordinate drain on his time and the frustration of pursuing a topic outside his normal areas of concentration, he enthusiastically sought ways to overcome my multiple, personal complicating factors. He met with me on Sundays for two years, entertained me in his home at odd times, and answered my calls and emails promptly in lieu of the many other items on his hectic schedule. Thanks Bill.

Contents

List of Figures

CHAPTER 1
Introduction

In this dissertation we give a formal methodology for verifying cryptographic protocols. To this end, we borrow a technique from the field of classical program verification called Weakest Precondition reasoning [DIJK76]. We first give a specification language to provide a formal syntax for protocols, and then utilize weakest preconditions to establish a formal semantics to the language, for the first time giving a definitive meaning to the actions of all principals in a protocol run, including intruders. The foundation on propositional calculus with uninterpreted function symbols allows this methodology to complement, rather than compete with, logical systems used to evaluate cryptographic protocols. The methodology is fully implemented in the DOS and UNIX environment in the C programming language. The result of this work is a methodology for evaluating the semantics of cryptographic protocols, complete with an implementation. The dissertation contains numerous examples of protocols evaluated using the methodology to illustrate the system's concepts and uses.

1.1 Network Security

From the ancient days, the need for effective communication across long distances was evident. Some of the most famous examples involve political leaders communicating with their subordinates, often regarding battle information in time of war. The Marathon gains its name and distance from the messenger that ran twenty six and two tenths miles to deliver the news of the arrival of a foreign force on friendly shores some three thousand years ago.

Julius Caesar recognized the importance and difficulty of controlling access to information passed across significant distances and went to great lengths to protect the privacy of the information he was communicating. He developed a method of encoding data

so that if the message were intercepted by an enemy, the enemy could not determine what the message said, but if delivered correctly, the intended recipient could easily read the message.

Nearly two thousand years after Caesar's venture with message encoding, security remains a concern of communicators of sensitive information. Though messengers are still an important form of communication, digital communication across sophisticated networks now dominates the volume of communication that occurs. Communications security and network security have begun to blend together and it is difficult to tell where one ends and the other begins. For this dissertation, we will consider communications security and network security to be synonymous. Our focus is on systems where messages are transmitted across some communications medium making transmissions vulnerable to interception and manipulation in route.

Privacy in communications has evolved to mean that only the intended recipient of messages can gain the message contents and their meaning, and that intruders can gain no meaningful information from the transmissions. Intruders are persons or computer processes whose intention is to interfere with the goal of communication. Routinely, intruders are outsiders to the communication system that break in to listen or otherwise compromise messages. Occasionally, intruders may be legitimate communication participants that exploit their legal status to compromise data.

Intruders take two forms: active and passive. Passive intruders are listening stations on the communications medium. For generality, passive intruders are assumed to be able to “hear” every message that is sent or received and are able to determine who originated each message and who each message is intended for. They do not manipulate data on the network and do not introduce their own transmissions.

Passive intruders compromise the security of the communication by analyzing

information gathered from the intercepted transmissions. If the data is readable by the intruder, the privacy of the data is directly compromised by the interception. Because it is very difficult to detect passive intrusion, techniques for protecting against it focus on prevention of listening. Traditionally, there are two approaches that are taken for this. The first approach is to physically protect data from interception. Whispering, armored car messengers, tempest computer and communications equipment, and sound-proof rooms are a few of the physical means used to guard against divulging sensitive data.

Encoding was devised to protect against compromise of data even though the data could not be physically protected from interception during transmission. Transmissions that are properly encoded cannot be read by an intruder even if the entire transmission is intercepted.

Passive intruders may also use sophisticated methods to analyze message traffic for volume, peak load, routing patterns, etc. using this information to infer facts and details that they are not supposed to know. For example, particularly heavy traffic volume on a military network may indicate to an intruder that some maneuver or attack is imminent. A routing shift from one location to another may signal a corresponding shift in emphasis of actions.

Passive intruders may also utilize covert channels, i. e. actions that are not routinely intended to communicate information, to send receive information in a way that was not intended by the designer of the communications system. The classical example of a covert channel is that of a clever, active intruder causing the disk accesses of a computer to create a meaningful pattern (such as a Morris Code message) that can be received by a passive intruder.

While passive intruders receive transmissions from the system but do not enter any messages onto the network, active intruders may receive and also transmit on the network

to accomplish a compromise. For generality, active intruders are assumed to be able to see and read all messages, intercept messages, modify messages and reenter them without inherent detection, and generate messages for entry into the network.

Active intruders offer a much more complex and challenging threat than passive intruders. Active intruders may make subtle changes to messages that may have a dramatic impact on the meaning of the message. An example we can all identify with deals with a bank's debit transaction. If an intruder can make a small change to the transaction by adding a zero just to the left of the decimal point of the debit amount, the meaning of the transaction can be changed significantly. If this account happens to be a corporate account and the original amount is in the millions, adding one zero just to the left of the decimal point can change the amount of the transaction by tens or hundreds of millions of dollars.

As this example illustrates, while privacy is the classic interest of communications and network security, concern for message integrity is equally important. For communication to be effective it is often critical to ensure that no message has been inappropriately modified and that each message is authentic.

In contrast to the defenses used against passive intruders, methods to combat active intruders center around detecting the attack, rather than preventing it. Our communications systems were designed to include as broad an audience as possible, often depending on broadcasting messages that are intended to be available to anyone that desires to receive the transmissions. Even when wire or cable technology is used, it is virtually impossible to prevent an intelligent intruder from gaining access to the media in a way that would allow them to receive transmitted data and messages. Thus, preventing intruders from intercepting messages, reintroducing old or modified messages, or introducing new messages is not possible.

For these reasons, battling active intruders focuses on detecting attacks. In the example above, if the bank receiving the debit can detect that an unauthorized change was made to the transaction, the transaction can be canceled and a request can be sent back to the originator asking for retransmission. Additionally, once an active intruder is detected, additional security measures may be taken to protect data and action can be taken to catch and stop the intruder from further mischief.

Authentication, the ability to guarantee the identity of principals in a communication, is critical to ensuring message integrity. In addition to making unauthorized changes to signed messages, an intruder may attack the authenticity of a message by forging signatures or by interrupting the sequence of a protocol. We say a message is authentic if we can verify that it originates from the person or process that the message claims or that the receiver expects. Signing messages and using passwords or pass-phrases are examples of techniques used to identify forged messages. The process of verifying the identification of the parties in a communication session is called authentication. We address authentication methods in a later chapter.

The final category of network security concerns is denial of service, i.e., preventing or substantially delaying incoming or outgoing message traffic from a principal. As an active attack, denial of service is very difficult to prevent. Among other things, service can be denied by sabotage of physical components, flooding, or manipulation of the controlling elements of the network. The focus of counter measures is to detect denial of service. Once denial of service is detected, action can be taken to restore service and protect against future similar attacks.

The general goal of network security is to provide principals in a communication system with a secure channel. A secure channel is a channel or conduit over which authenticated, private communications may occur. Cryptography allows the conduit to be logically secure in the sense that the physical medium may be vulnerable to intrusion, but

physical intrusion cannot compromise the security of the communication.

1.2 Cryptography

Cryptography is the science of making and using codes and ciphers. A goal of cryptography is to provide secrecy of data transmitted between sender and receiver for some period of time. If perfectly implemented, only the intended recipient can read an encrypted message. Encryption algorithms are designed so that decoding encrypted messages to divulge their meaning is expected to take sufficiently long so that any data compromised will be useless to the intruder. As cryptography grew into a broad field aimed at finding strong encoding algorithms, a corresponding field of study emerged aimed at attacking those algorithms by attempting to decode encrypted messages. This field is called cryptanalysis.

1.2.1 Data encryption.

Julius Caesar became the first documented user of encryption when he devised a method to protect the privacy of his communications. The coding scheme, or cipher, he devised is now known as a substitution cipher and involved “rotating” the letters of the message three letters “forward”. For example, FDW is the “cipher text” for the word CAT when CAT is encoded using Caesar's method. This simple procedure was effective in preventing unauthorized persons from reading written messages that were captured in combat.

Today, the applications and techniques for encoding (now called encrypting) data are far more complex than the famous Caesar Cipher. Ciphers were perpetually updated and improved to combat cryptanalysis. An improvement of the Caesar Cipher involves utilization of a piece of information called a key, known only to the two communications principals, to make the cipher harder to break. The fundamental characteristic the key must have is that it should be possible to decrypt an encrypted value using the same key

that was used for encryption to return the original encrypted information. Symbolically, if we assume e[X]k to mean, “the encryption of value X under key k”, and d[X]k has the corresponding meaning for decryption, then the fundamental characteristic of a key is:

d[e[X]k]k == X

The introduction of an encryption key eliminated reliance on the secrecy of the encryption algorithm for the strength of the cipher. By using a key, the encryption algorithm may be made public, yet encrypted data will remain private between owners of the shared key.

The obvious key first selected for the Caesar Cipher is the “rotation number”, which for the original cipher was three. By agreeing to a number known only to them, the communicators made the cryptanalyst job more difficult. A further advance is to use a twenty six character key so no “rotation” is needed and the population of valid keys is much larger.

While the addition of a key and other enhancements clearly make breaking the cipher harder, cryptanalysis research found techniques that allow easy decrypting of encrypted data (cipher text) using any of the key methods we described for a substitution cipher. Hence, a second method of encryption evolved which involved partitioning the data into fixed size blocks and shifting the letters in some predetermined pattern. This is called a transposition cipher. The encryption key in this scheme is the order of the letters in each block, which the recipient can use to easily decrypt any message encrypted using this scheme. For example, if we use a five byte block transposition cipher with a key of “52413”, the phrase

“attack_at_dawn”

would be encoded:

ctaat__tka_andw

Unfortunately (at least for cryptologists), transposition ciphers are also vulnerable to attack by a knowledgeable, systematic intruder.

Another paradigm for encrypting data is using a technique called codes. Codes rely on assigning a number or random pattern of characters to represent every word and phrase used in communication. Using a code book, the transmitter encodes the message into a stream of characters that is meaningless to anyone without the code book. The receiver utilizes the code book to translate the message back into its original form.

Substitution ciphers, transposition ciphers and codes each have varying advantages and disadvantages. Most importantly, each is vulnerable to cryptanalysis. However, research aimed at combining pairs of these techniques resulted in encryption systems that are very difficult to break. These techniques involve re-encrypting already encrypted data. One of the most famous instances of combining ciphers utilized the substitution cipher and codes. It was this scheme, called Super Encrypting, that the Japanese used in World War II. Their belief that Super Encryption was unbreakable resulted in great damage to their war effort when the Allies effectively cryptanalyzed the Japanese Super Encryption and were able to gain very valuable information about Japanese plans which the Japanese believed were private. While Super Encryption provided strong protection, it was not perfect.

With the major advances in automatic computing in the past fifty years, cryptography also evolved dramatically. By coding encryption routines into a computer, combinations of encoding techniques are easily implemented and many “rounds” of encryption can occur, that is, data can be encrypted and re-encrypted many times over very quickly in order to strengthen the code against cryptanalysis. Of course, cryptanalysis also benefited greatly from the computer explosion, leaving the battle between the cryptographer and the

cryptanalyst as intense as ever.

Research guided by the National Security Agency (NSA) resulted in establishment of the Data Encryption Standard (DES) [DES77]. DES is a published an encryption algorithm that communicating parties can use to encrypt and decrypt data. DES uses repeated rounds of substitution and transposition ciphers which lend themselves easily to computerization and can be used to encrypt large amounts of data quickly on general purpose computers. While there were early concerns about the strength of the DES algorithm, experts now agree that the DES design results in a cipher that is very difficult to break. Other variants of DES are still emerging which address DES's one weakness, the small size of the key-space (56 bits), making data encrypted using DES vulnerable to computerized "brute force" attacks which effectively conducting an exhaustive search of the key space to find the key.

An obvious characteristic of private key encryption systems is that each participant must know the same key that the other participant knows and that the shared key must be private to the two of them. Because of the "symmetrical" nature of these keys, private key systems are also known as symmetric key systems. In their seminal work [DH76], Diffie and Hellman offered an alternative to symmetric key systems. Their "asymmetric" key paradigm allows a principal to derive a pair of keys which are inverses of one another. The keys are inverses in the sense that data encrypted under one key of the pair can only be decrypted using the other key (its inverse).

The primary implementation of asymmetric key systems is in what is known as Public Key systems. In such a system, each principal publishes one member of its asymmetric key pair. This becomes the member's public key. The public key inverse is maintained and kept private by the principal. When one principal desires to communicate privately with another, the originator looks up the public key of the intended recipient, applies the encryption algorithm to the data to be passed, and sends the cipher text to the

recipient. The recipient utilizes the public key inverse, (sometimes called the “private key”) to transform the cipher text into its original form. Because only the owner of the public key knows the public key inverse and if the encryption algorithm is strong, no one else can decrypt the message.

There has been significant research regarding development of encryption algorithms since cryptography was born. However, to date only one perfect encryption system has been found. To be perfect, the cipher text must not give an intruder any information they can use to divulge the contents of the message or the message key. The one perfect encryption system, a symmetric key system called One-Time Key, has the drawback that the length of the key is identical to the length of the data to be transmitted. As we noted above, the DES encryption algorithm is believed by to be a strong encryption system. We say that an encryption algorithm is strong if the difficulty of decrypting the cipher text via cryptanalysis can be accurately computed in terms of the amount of computer resources that would be required to divulge the original plain text or key and that those resources are very high.

Many attempts have been made to devise strong asymmetric key systems. Some were based on mathematical problems that are known to be computationally intractable, such as the Knapsack Problem. Others have focused on hard problems in number theory such as discrete logarithms [ELGAM88]. For most of these mechanisms, weaknesses known as buckaroos, exist that allow intruders to compromise the data or the key with much less effort than base problem demands.

In [RSA78], Rivest, Shamir, and Adleman proposed an asymmetric key algorithm called RSA, based on the difficulty of factoring large numbers. No back-door has been found in their RSA public key encryption algorithm and while it cannot be proven secure, most experts agree that the RSA algorithm does provide strong encryption. The RSA algorithm now dominates the cryptography world as the public key algorithm of choice.

Research continues to seek strong, efficient encryption algorithms and to find flaws and back-doors in existing algorithms that are believed to be strong. However, even if a "perfect" encryption algorithm is found, encryption alone is not sufficient to protect communication. Encrypted messages must be combined with a handshaking interaction between the communicating parties called a Cryptographic Protocol. We examine cryptographic protocols in the following sections.

1.2.2 Cryptographic Protocols

Protocols are rules that govern interactions between communicating parties and are fundamental to the implementation of communications systems. Protocols are used to allow parties in a communication session to remotely reach agreement in some area. Cryptographic protocols are security related interactions that support the same objective: gaining agreement regarding some important topic. In security related protocols, two common agreements to be reached are for authentication and key distribution.

Authentication, the act of determining the identity of a principal, is fundamental to any security system. Because of this, and because of its complexity and potential for error, authentication methods have been the topic of intense research. Most authentication schemes rely on some interaction, or challenge and response, between the authenticator and the one to be identified. These exchanges are called authentication protocols.

Authentication protocols may be as simple as the authenticator offering a challenge to someone seeking physical access, such as the verbal command "Halt", to which the one to be identified would respond with some predefined password. If this password is sufficiently hard to guess and is well managed so that only the two principals in the exchange know it, success of this protocol offers reasonably sound evidence that the respondent is actually who the authenticator believes they are. If, however, an intruder is able to overhear the interaction and if the password is reused or is used to identify more

than one individual, the intruder may use the overheard password to defeat the protocol.

Research in authentication protocols is aimed at identifying the potential threats to authentication and devising interactions that cannot be broken, i.e., successful execution of the authentication protocol does indeed provide strong evidence as to the identity of the subject(s). Cryptography is very useful for this purpose, which we will see in greater detail later.

There is no doubt that the introduction of keys into cryptographic systems was a revolutionary change that gave cryptographers a decided advantage over intruders, at least for a while. It also introduced the problem of key distribution into cryptographic systems. A characteristic of key-dependent systems is that the longer a key is used, the less secure are the communications using that key. Keys must be changed regularly and often. Privately distributing keys between remote principals is a very difficult, now classical, problem termed the Key Distribution problem [MEAD92]. Like authentication, key distribution is routinely accomplished by a cryptographic protocol.

As noted earlier, in order to address the network security problems of privacy, integrity, authentication, key distribution, etc., a strong cryptographic algorithm must be combined with a valid cryptographic protocol that establishes the rules governing interactions between participants (principals) in the communication. Only when combined with a valid protocol can cryptography provide the principals with a secure channel.

Cryptographic protocols are routinely represented as action lists describing the alternate transmission and receipt of messages between principals. The steps of the action list specify the contents of each message and the encryption and decryption operations utilized to protect and divulge the message meaning. The content and sequence of messages in the protocol agree to some pre-defined format and sequence. A successful run of the protocol may be seen as a serial trace of the steps of the protocol in the specified sequence.

It is the requirement for serial sequencing that is of interest here. The de facto serialization method that provides the sequencing in protocol runs is the blocking nature of the send and receive statements. This means that in a protocol run, a principal may not receive a message of a certain format until after a message of that format has been generated and transmitted. While it is clear that a receive cannot be legally accomplished unless a message of the specified format has been sent, it is not clear that the appearance of a message of the correct format means the protocol has executed in the correct sequence.

Figure 1.1 is an example of a cryptographic protocol; in fact, one of the earliest published ones. The Needham and Schroeder Private Key Protocol played a large part in the field of cryptographic protocol verification. The meaning of the notation is intuitive; the arrows indicate transmission and receipt of messages. The items after the colon for each step reflect the data to be transmitted and the braces signify encryption of the data within the braces. The identifier immediately after each right brace is the key used for encryption. The protocol is expressed in an hoc standard cryptographic protocol pseudocode language with the messages presented in the order they are expected to be executed.

Needham and Schroeder Private Key Protocol

A->S: A,B,na
S->A: $\{na,B,kab,\{kab,A\}^{kbs}\}^{kas}$
A->B: $\{kab,A\}^{kbs}$
B->A: $\{nb\}^{kab}$
A->B: $\{nb-1\}^{kab}$

Figure 1.1

In order to illustrate the meaning of the pseudocode notation, consider the given protocol. In the first step, principal A indicates to the central authentication server that A desires to initiate a secure communication with principal B. A includes with this message a random number called a nonce [NS78] to be used to guarantee the freshness of a later

message. Nonces are, as Webster defines in the new world dictionary, "... created only once for a special occasion". They are selected randomly so that they are unpredictable so any message received containing a specific nonce is assumed to be a response to the original message containing the nonce.

In the second step, S provides A a key to be used in a communication session between A and B, appropriately named kab. Also included in the second message is an encrypted component that A cannot read containing the key kab and the identity of A encrypted under the key shared between B and S (kbs). This encrypted catenation will be forwarded by A to B in the next step. The entire second message is encrypted under the private key kas that A shares with S. A uses kas to decrypt the message and obtain the session key kab, nonce na, and the encrypted message to be relayed to B. Before executing the third step, A verifies the currency of nonce na by comparing the representation of na that was received from the second step with the value na that A transmitted in the first message.

In the third step of this protocol, A forwards the message generated by S to B. B uses the key kbs that B shares with S to decrypt the message and obtain the key kab and the identity of the originator, A. In step four, B forwards a new nonce to A encrypted under the session key kab. When A receives the message sent in step 4 and uses kab to decrypt the nonce from B, A then believes B has the key and is ready to communicate because A believes that only B could know key kab.

In the last step, A modifies the nonce from B slightly, re-encrypts the new value and sends it to B. Because A modifies the nonce from B in a predetermined way and since B believes that only A could have the key kab, when B decrypts the modified nonce and compares it to the expected response, B believes that A has the key and is ready to communicate.

1.3 Cryptographic Protocol Verification

The Needham and Schroeder [NS78] protocol given above appears to be quite simple. As shown, it contains only five messages, involves a total of seven data items, and requires only five encryption and five decryption operations. Routinely, cryptographic protocols are short, contain few data items, and require less than twenty encrypt and decrypt operations. While this may lead one to believe these protocols are easily and effectively verified manually, research proved that this is not the case. In fact, the Needham and Schroeder Private Key protocol itself has a flaw that we will investigate in some detail later.

There are many cryptographic protocols in the literature, [NEED78], [OTWY87], [DS81], and [DOL83] for example, which were designed to provide the coordination necessary for effective, secure communication. These protocols, like the encoding schemes they support, may contain subtle flaws that could compromise the privacy of messages even if the cryptographic technique itself is sound. Because of the complexity and potential subtlety of the flaws, manual verification of these protocols is not sufficiently rigorous; formalized mechanisms are required.

1.3.1 The Need for Cryptographic Protocol Verification

In order to illustrate the need for cryptographic protocol verification, consider a simple, one-step protocol example of a stock brokerage firm using cryptography to communicate from its main office. In this example, each broker shares their own private key with the main office. On a given day, the main office (say M) may desire that a broker (B) buy or sell a particular stock (S). The first thing in the morning, the main office sends a message encrypted under B's private key (kmb) instructing B to buy:

M: =>B(e[“Buy $10,000 of S”]kmb);

The notation is taken from our Cryptographic Protocol Analysis Language (CPAL)

which we describe in detail in Chapter 3. `M' is the party taking the action, '=>' represents the send operation, `B' is the destination of the message, and the literal message `Buy $10,000 of S' is encrypted under key `kmb' before transmission.

The execution of this protocol is also simple. When B receives the message, the common key `kmb' is used to decrypt the message and the action described in the message is taken. But what if an astute adversary is listening on the net and recording the encrypted messages as they occur? By watching the subsequent actions of the broker on the floor, the adversary could reasonably predict what the message said, though after the fact. If this adversary could subsequently intercept messages from the main office and substitute recorded messages either at random or with some selectivity based on his own insight, he could potentially do damage to the brokerage firm without ever compromising an encryption key or penetrating the encryption algorithm.

A second problem with the example protocol is that once the intruder knows what the message means, the cipher text can be matched to the plain text message that generated it. This is called “known plaintext” which is shown to give an intruder a decided advantage in being able to cryptanalyze the encryption algorithm.

This example clearly illustrates that authenticating users across a network is not as straight forward as one might think. Current research describes the difficulty with the example protocol as that of freshness. There is no mechanism in the protocol to prevent an intruder from reusing old messages. For even simple protocols, there are many factors that effect the ability of the protocol to meet its goals. The challenge to the intruder is to envision actions that the protocol developer did not see in order to compromise a key, divulge private data, interfere with message integrity, or deny service to a valid principal.

The difficulty with message freshness illustrates another problem with protocol verification. There may be many different ways to accomplish the same objective within a

protocol. Take the issue of message freshness for example. Over the years, three different mechanisms evolved to signify freshness of messages in a protocol: time stamps, sequence numbers, and nonces (random numbers). Each has different characteristics, different strengths and weaknesses. For example, combining the message with a nonce eliminates the known plaintext problem while the other two do not. Each technique is also implemented in different ways, yet they can each be used to accomplish the same freshness goals.

The problem quickly becomes even more complicated as the varied goals and environments of cryptographic protocols are considered. For example, cryptographic protocols are frequently used for authentication such as in the broker example just presented. This example illustrated one-way authentication. Only the broker needed to be sure that the main office sent the message. In another environment, it may be important for two-way authentication to occur, that is, for each party in the communication to know who is on the other end. It may also be important to be able to prove at a later date that a particular communication came from its signed originator. This can be accomplished by utilizing a third variety of authentication protocol called digital signatures.

We discussed three types of authentication that can be accomplished using cryptographic protocols. What is more, cryptographic protocols are not only used for authentication. They are also utilized for key distribution, to ensure integrity of messages, to detect denial of service and for variations on and combinations of all the above objectives.

Goals of cryptographic protocols are further complicated by the varied security environments existing in distributed systems. The simple illustration we used above is an example of a two party protocol, where the only valid participants are the principals requiring the data transfer. Much of the work described in the literature focused on a three party security environment that includes an authentication server [DS81], [NEED78], [OTWY87]. An authentication server is a principal that assists other principals in estab-

lishing secure communication channels. The traditional function of an authentication server is to assist principals in authentication by providing such simple tasks as directory services for public keys and in creation and distribution of valid cryptographic keys.

Use of an authentication server creates several security problems itself. In order to accomplish its functions, the authentication server must have a valid shared key with each distributed principal. Distribution and protection of these keys is a security problem of essentially the same kind they are intended to solve. Additionally, having a central authority through which principals must communicate is contrary to the philosophy of distributed systems. In this scenario, the authentication server is a single point of failure of the system and is also a potential system bottleneck, which are two critical flaws that distributed systems are supposed to avoid.

There are cryptographic environments that use an “x-party” authentication server model. These systems are supported by protocols that distribute the security functions of authentication and key creation to several distributed authentication servers [GONG93]. Cryptographic protocols in an x-party authentication server environment differ significantly from those in two and three party environments.

A further complicating factor in evaluating whether or not cryptographic protocols meet their goals is the cryptographic technique utilized. Protocols for symmetric and asymmetric key systems differ in both the assumptions that can be made, and in the steps required to accomplish goals. Private, or shared key systems, require that any two principals desiring to have secure communication must share a single private key agreed to before the protocol begins. The key is used by the sender for encryption and by the receiver for decryption. Public key systems, on the other hand, have separate keys for encryption and decryption. Each station's public key is generally distributed universally to all principals and is retrieved via some trusted library-type function. Stations wishing to communicate secretly with another principal encrypt the intended message using that prin-

cipal's public key and send it across the network. Anyone listening to the network can receive the message, but since only the holder of the inverse key can decrypt the message, the communication is private. The original message may contain a shared key to be used for the session, or the recipient may respond using the public key of the originator. Authentication and other cryptographic protocols in public and private key environments may differ significantly.

1.3.2 The Origin of Cryptographic Protocol Verification

While the simple broker example we gave in the last section illustrates the need for cryptographic protocol verification, the foundation for the research presented here lies in the 1978 paper by Needham and Schroeder [NEED78]. In that paper, the authors present a group of protocols for systems with authentication servers for shared and public key authentication and signatures. These protocols effectively highlight many of the important cryptographic protocol issues and are frequently used in current literature for purposes of illustration. Needham and Schroeder propose three protocols: one for authentication using private key systems, another for authentication using public key systems, and a signature protocol. Each of these protocols are written for systems which utilize authentication servers. Their private key protocol was given in Figure 1.1.

In 1981, Denning and Sacco [DS81] utilized the Needham and Schroeder private key protocol to illustrate the problem of replay attacks. They showed that if an intruder is recording messages and can compromise a session key over a period of time, then the intruder can begin an erroneous communication session by replaying the third message from the previous secure protocol run which used the compromised key. The difficulty with the protocol is centered around the third message. Not coincidentally, the third message does not contain a nonce (i.e. a random number used to guarantee freshness of a response). Needham and Schroeder acknowledge this weakness in the protocol and introduce a minor modification inserting a timing mechanism into the protocol to prevent

replay attacks in [NEED87].

In [OTWR87] Otway and Rees address the timing problem by slightly changing the sequence of messages and by including a session identifier in each message. Their protocol is given in Figure 1.2. Otway and Rees protect against replay attack by including a session identification value (C) in each message. The message sequence is similar to the Needham and Schroeder, except the only principal that communicates with the authentication server is the target principal of the communication, B. In step 2, B forwards a message from the protocol originator to the authentication server, and in step 4, from the authentication server back to the protocol originator.

Otway and Rees Private Key Protocol

A->B: (C,A,B,{Na,C,A,B}^{kas});
B->S: (C,src,dst,{Na,C,A,B}^{kas},{Nb,C,A,B}^{kbs})
S ->:B (C, {Na,kab}^{kas},{[Nb,kab}^{kbs})
B->:A (C,{Na,kab}^{kas})

Figure 1.2

In [NEED78], Needham and Schroeder conclude, "... [cryptographic] protocols such as those developed here are prone to extremely subtle errors that are unlikely to be detected in normal operation. The need for techniques to verify the correctness of such protocols is great...". It is ironic that this point is illustrated by the flaw detected in their own protocol. For years, computer programmers relied on layers of testing to address the similar problem of finding errors in programs. Unfortunately, testing of protocols cannot provide a complete answer because of the cleverness of intruders. Testing looks for errors, yet failure to find errors does not mean that errors do not exist [DIJK76]. Accordingly, extensive effort has been focused upon finding methods to verify cryptographic protocols formally.

1.4 Problems with Protocol Specification

Fundamental to determining if a protocol meets its goals is having a clear accurate description of the steps of the protocol. Without a suitable specification language and evaluation methodology in place, establishing such a clear specification of the protocol is unlikely. There is a preponderance of research regarding specification languages from the traditional programming languages field. In [GUMB82] Gumb identified ten characteristics which are required for any specification language for a formal verification mechanism. This section addresses some of the shortcomings of the specification step of cryptographic protocol verification.

1.4.1 Languages for Cryptographic Protocol Specification

A difficulty in evaluating cryptographic protocols results from the pseudocode languages routinely used for specification. These pseudocode languages are frequently seen in the literature, but have no agreed formal definition. Hence, the intent of a principal's actions in the protocol can only be truly known by the protocol specifier. An illustration of this idea is that these languages couple important actions, emphasizing expressiveness and ease of coding at the expense of rigor. For example, the following statement may appear in a pseudocode protocol expression:

$$A \rightarrow B: \{X, Y\}^{kab} \qquad (1.0)$$

This indicates that principal A is sending to principal B the messages X and Y encrypted under some common encryption algorithm and under the common private key kab. Because these pseudocodes evolved for the purpose of specification, they represent the protocols in a high level of abstraction. As illustrated above, a single send operation represents numerous actions that the principals must take. One example is that the receive action is implicit in the send operation. While it is intuitively clear that when a principal A sends a message M to principal B, that B is expected to receive M, omitting the receive operation prevents us from expressing actions that may occur between the originator's

send and the destination's receive actions.

Similarly, the encryption and decryption actions are part of the send operator. Consequently, it is difficult to determine if the sending principal encrypted the message or if the principal is forwarding a message that was previously encrypted by another principal. In current practice, context is required to make this determination. Assigning responsibility for all actions to principals is greatly complicated by the broad spectrum of actions represented by the pseudocode send operation.

While the coupling of operations in the pseudocode languages can be cleared up largely with additional syntax, the omissions of the pseudocode languages cannot. The data items in the messages must have meaning to the principals. The pseudocode languages provide no mechanisms for specifying what the data items themselves mean, or even who originated the message. Moreover, many of the protocols rely on comparing data items sent in one step and received in a later step, yet the languages do not provide mechanisms for depicting relationships between data items. Conditions for continuation from step to step and conditions for the success (or failure) of the protocol are either left to intuition or specified in a narrative accompanying the protocol. Worse yet, many protocol specifiers introduce local conventions or utilize special naming practices to add information to the protocol specification that not all readers may understand and thereby may miss information that is important to the protocol operation.

Another shortcoming of the pseudocode languages used to specify cryptographic protocols is that they do not provide mechanisms to assist in the determination of valid assumptions, such as preexisting private keys between participants. There is also a need for mechanisms for specifying protocol goals, like “A believes k is a good key”. In order to mechanically evaluate the success of the protocols, both the assumptions and goals must be expressed in a form suitable for this evaluation. We propose a language in which assumptions may be formalized in terms of actions of participants. We believe this

explicit declaration of actions formalizes and simplifies the designation of assumptions. In our language goals are expressed as predicates that a participant desires to be true at the specified point in the protocol.

1.4.2 Protocols, Logical Systems, and Step Sequencing

Protocols are routinely specified as a serial list of send and receive statements, or steps. Recently, it became clear that while the specification of the protocol may be serial, any method of protocol verification must consider parallel transmission of messages. We will investigate this situation more fully later in the dissertation.

Many efforts recently have focused on formalizing cryptographic protocol verification by developing logical systems that capture the essence of protocol meanings and vulnerabilities. The central theme of these logical systems is to examine the beliefs of the participants given a set of assumptions, a set of rules of inference, and a protocol expressed in terms of the impact of a protocol step on a participant's beliefs.

While logical systems add a great deal to the field of cryptographic protocol verification, they continue to lack the formality they are intended to provide. To use the logical systems, the protocols must be stated in the language of the logic, while protocols are specified and accomplished by actions of participants. This leads to the problem of idealization we discuss in the next section. Other fundamental difficulties have also been identified.

Snekkenes [SNEK91] identifies the sequencing of the protocol statements in the specification as one aspect of protocol verification that is particularly difficult for logical systems to represent. Since non-temporal logical systems cannot enforce strict sequencing on the protocol steps, the logical evaluation considers actions as timeless. In truth, the sequence in which the actions occur are essential to the security of the protocol. If there exists an ordering of the protocol steps that does produce a secure protocol, then the proto-

col will pass the logical evaluation regardless of whether the steps are correctly ordered in the evaluated protocol [SNEK91].

In his critique of the logic by Burrows, Abadi, and Needham [BAN90], Nessett [NESS90] argues that the inability to detect all faults is a critical flaw, since the positive evaluation may give a user a false confidence in the flawed protocol. While the soundness (accuracy when flaws are detected) of BAN is important, completeness (ability to detect all flaws) is also necessary for an effective verification tool. We discuss logical protocol verification in greater detail in the next chapter.

1.5 A Solution From Formal Semantics of Programming Languages

While Dijkstra discredited testing as a means to ensure proper program execution [DIJK76], others continued investigation into methods of proving "program correctness" [HOAR69], [HOAR78], [WULF81]. These methods were formal in that the semantics of the programs took their meaning directly from the syntax (or form) of specification. Specifically, formal languages were developed that were expressive enough to describe the needed functionality, then a formal semantics was given to the language. By formally defining the syntax of the language to be used for specifying programs and then attaching a formal semantics to the language, a formal definition of the meaning of any program in the language can be derived. One can then use the formal definition to determine whether or not the program means what we want it to mean.

In order to give meaning to programs, Hoare [HOAR69] gave each program structure a precondition/postcondition definition. Using these definitions, Hoare could generate the definition of the segment S in terms of precondition P and postcondition Q. The program proof began with the claim: "If condition P is true before segment S runs, and if S runs to completion, then condition Q will hold afterwards." Condition P would then be a verification condition, a condition whose truth guarantees the truth of the original claim

(Q). Once the verification condition was generated, all that would be left was to show that the assumptions imply the verification condition.

This is precisely the approach we take with cryptographic protocol verification. Cryptographic protocols are programs; a special case of programs that are short and do not require constructs even as sophisticated as a procedure call. We began by defining a formal language that closely mirrors several of the pseudocodes routinely used in the literature. The language we call the Cryptographic Protocol Analysis Language (CPAL) is very simple. Besides having no procedure or function call constructs, it provides no looping structure, and none of the powerful definitive features of modern programming languages. CPAL's simplicity and similarity to pseudocodes already in use combine to make the language easy to use for writing and carefully defining cryptographic protocols. Value is added to CPAL by including operators to allow protocol authors to explicitly list local and global goals (ASSERT/GASSERT), assumptions (ASSUME), and random number generation (NEW). A simple if-then-else construct is also provided. More detail of CPAL is given in [YW93] and in the next chapter.

The tool we chose for developing the formal semantics for CPAL is the weakest precondition reasoning of Dijkstra [DIJK76]. Two characteristics of this tool are attractive as the basis of our formal semantic mechanism. First, we are able to automate the process of generating the verification condition and second, using weakest precondition definitions we are able to enforce strict sequencing on the steps in a protocol. We established CPAL's formal semantics by developing the weakest precondition for each CPAL statement. To verify a protocol, we begin by encoding the protocol in CPAL. Then, using the weakest precondition definition for each statement, we mechanically derive a “verification condition” for the protocol. A verification condition is a condition whose successful proof will guarantee the success of the goals expressed in the protocol. The final step in our verification process is to prove the verification condition.

Fortunately, proof of the verification condition is rarely complex, mostly involving substitution of simple identities such as “A and A reduces to A”. Assumptions are also applied to the verification condition in this step. Greater detail of the formal semantics for CPAL is given in Chapter 5 and examples of the application of this process are provided in Chapter 6.

CHAPTER 2

Previous Research Regarding Cryptographic Protocol Verification

Extensive research regarding the Cryptographic Protocol Verification (CPV) problem followed the Needham and Schroeder paper. In this chapter we will show the breadth and depth of this research, and illustrate why we believe no suitable solution has been found.

In [BAN88] and an updated version of the paper [BAN90] Burrows, Abadi, and Needham made a giant step forward in resolving the CPV problem. The Logic of Authentication (hereafter called BAN Logic) they proposed achieved widespread acceptance as the de facto standard system for CPV, and is the target of enhancements and supplements by various authors [GNY90], [AT91], [SYV93a], [SYVOOR94]. BAN Logic and two other logical systems based on BAN Logic are described later in this chapter.

While BAN Logic is widely accepted, Nessett's paper [NESS90] demonstrated a weakness in the use of BAN Logic for protocol verification. This triggered broad and often spirited discussions regarding cryptographic protocol goals, environments, and other categorizations [BAN90b], [SYV91], [SNEK91]. Moreover, in much the same way as Denning and Sacco [DS81] began the surge for examining cryptographic protocols, [NESS90] challenged the very mechanisms used to evaluate protocols and triggered intense research in suitable CPV mechanisms [BAN90b], [SYV91], [SNEK91]. As a result, CPV mechanisms are now subjected to the same level of scrutiny as the protocols they evaluate. Several fundamentally different mechanisms evolved in the research.

In [MEA92], Meadows identifies the four major categories of protocol verification methodologies as:

1 - Use of specification and verification tools developed for software evaluation.

2 - Testing tools based on expert systems which evaluate different scenarios.

3 - Model the protocol requirements formally in a logic of knowledge or belief.

4 - Model the protocol formally based on the algebraic term-rewriting properties of cryptographic systems.

Examples of the first, second and fourth of these are compared in [KMM93] and described in detail elsewhere in the literature. BAN Logic is an example of the third of these categories. Kemmerer proposes a method which spans the first and second categories in [KEM89]. The method we propose later in this paper spans the first, second, and third categories.

2.1 Logical Systems for Evaluating Cryptographic Protocols

As we described earlier, a logic is a set of constructs, or logical language, combined with a set of axioms and a set of rules of inference that utilize those constructs. We discuss the specifics of several logical systems later in this chapter. We now briefly discuss the methodology for using these logical systems. Generally, logical systems designed to verify cryptographic protocols operate in the same three steps:

1 - The protocol is translated from its procedural notation into a formula notation, i.e., the language of the logic to be used. This process is called *idealization*.

2 - Assumptions are made about the state and environment of the system.

3 - Facts about the protocol session are derived from the assumptions using the axioms and rules of inference

The benefit of evaluation by systems of logic lies in the last step. Logical analysis based on an axiomatized language allows a logician to mechanically prove theses specified within the language. Unfortunately, the final step is critically dependent upon the first two steps, which are much less effective in accomplishing their purpose.

Idealization involves translating the meaning of the protocol specification from its

description of the principal's actions expressed in a procedural notation, to a corresponding description of the principal's beliefs and intentions given in a formula, or logical language. Accomplishing this translation is more complex than one might expect. We do not know of any mechanism yet developed that allows translation from the specified actions of the principals to the beliefs that those actions reflect in a formal, mechanical way. This is largely because global knowledge of the intent of the protocol is required to make this transformation. According to Burrows, et al. "... the idealized form of each message cannot be determined by looking at a single protocol step by itself. Only knowledge of the entire protocol can determine the essential logical contents of the message" [BAN89]. The authors then provide three non-mechanical guidelines for making the idealization transformation:

1 - "... a real message m can be interpreted as a formula X if whenever the recipient gets m he may deduct that the sender must have believed X when he sent m."

2 - " Real nonces are transformed into arbitrary new formulas; throughout, we assume that the sender believes these formulas."

3 - "... for the sake of soundness, we always want to guarantee that each principal believes the formulas that he generates as messages."

Idealization is further complicated by the informality of pseudocode languages used for cryptographic specification. If the interpreter of the protocol accurately judges the meaning of the protocol action, this still does not guarantee that the idealization will accurately reflect what the specifier intended. In fact, it is likely that two different idealizations of the same protocol may exist even though there is no ambiguity in the actions of the principals. We discuss the difficulties of protocol specification in greater detail in the next chapter.

BAN Logic was among the first of many logical systems designed for protocol verification. With its small number of constructs and simple, concise rules of inference,

BAN Logic became the foundation for expansion by authors attempting to extend and improve the BAN Logic results. Among others that followed were [AT91], [BEIB90], [GLAS88], [GS91], [GNY90], [KG91], [MB93], [MOS89], [RANG88], [SNEK91], [SYV90], [SYV93a], and [vO93]. Several of these are described later in this section.

The logical systems that evolved differ in important ways. For instance, some of these logical systems are founded upon knowledge (epistemic modality: A knows P) while others are founded on belief (doxastic modality: A believes P). The primary difference between logical systems of knowledge and logical systems of belief is that knowledge based logical systems have an axiom of the following form: "If you know P, then P is true" (Syverson calls this Axiom T in [SYV91]). Belief systems do not have Axiom T. In belief based systems, belief in P says nothing about the truth or falsity of P.

In [SYV91], Syverson differentiated between evaluation mechanisms based on trust and those based on security. He suggests that systems concerned with trust consider the meaning of a protocol in an environment of exclusively trusted participants. Hence, the evaluator *trusts* that each participant will follow the rules of the protocol. On the other hand, mechanisms concerned with security consider the effects that dishonest principals (or malicious intruders) may have on the outcome of the protocol by not following the protocol rules. Syverson further argues that logical systems of belief are not suited for reasoning about security, but can be used to evaluate trust in a protocol, while security as well as trust of cryptographic protocols can be evaluated with logical systems of knowledge. Later research [SYV92] suggests that knowledge and belief based logical systems have equivalent reasoning power. We discuss logical systems of belief and logical systems of knowledge in more detail later.

Many of the enhancements to BAN Logic [AT91], [GS91], [GNY90], [KG91], [MB93], [SNEK91], [SYV90], [SYV93a], [vO93] were designed to deal with the step sequencing problem identified by Snekkenes [SNEK91]. Temporal modalities are given

significant attention, largely for the purpose of evaluating the freshness of messages. Abadi and Tuttle [AT91] introduce a "forwarded X" construct for distinguishing newly generated messages from forwarded messages. The second message in the Otway and Rees protocol given in Figure 1.2 is an example of a component of a message that is not generated by the originating party to the destination, but is forwarded from some third party. Abadi and Tuttle also introduce a "P says X" construct to amplify the BAN Logic construct P said X. P says X indicates that principal P said message X in some protocol run.

Gong, Needham, and Yahalom [GNY90] also provide similar constructs for forwarding messages and provide a "P once said X" to indicate the freshness of X is not yet known. They further provide a construct which allow them to distinguish between messages that one recognizes and those that one believes.

Kailor and Gligor [KG91] present a methodology based on ordered knowledge sets, allowing principals to accumulate knowledge as the protocol progresses. A key mechanism in their syntax is the requirement to include the message round of each message in the protocol specification. Syverson [SYV93A] adds pure [past] temporal logic operators to the Abadi and Tuttle logic. He demonstrates that by using the Abadi and Tuttle logic with these operators, causal consistency attack flaws in cryptographic protocols may be demonstrated.

While the efforts we described and others not described (or even noted) have contributed significantly to the understanding of cryptographic protocol complexity, no method has yet achieved widespread acceptance and use in verifying cryptographic protocols.

In the now classic paper [SYV91] Syverson brilliantly lays out many of the important arguments surrounding logical systems used for evaluation of cryptographic protocols

and gives a compelling argument for associating an independently motivated formal semantics with the logical syntax. He points out that while BAN Logic adds formality to the cryptographic protocol verification process, it is not fully formal. He reminds us that the difference between rigor and formality is structure and that no amount of rigor can make a method formal. Formality comes only if the method is based exclusively on form, in our case, the syntax of the specification.

In order to attach semantics to the logical systems, different modalities (e.g. temporality, possibility, necessity) of propositions are considered. The possible worlds scenario [RANG88], used extensively for logical systems geared to evaluating cryptographic protocols, allows the logician to reason more effectively about the real world by emphasizing possibility, impossibility, and necessity [BAN90], [SYV91]. Briefly, a possible world is a set of propositions that represent the way the world may be. For example, there may be a possible world in which there is gravity and another in which there is no gravity. We can determine which possible world(s) a principal may be in by comparing their knowledge set with the facts about the possible world, i.e. a principal that knows that there is gravity cannot be in a possible world in which there is no gravity. Each belief that is added to the principal's knowledge set reduces the number of possible worlds that the principal could be in.

Syverson [SYV91] illustrates the application of possible worlds using the obviously insecure protocol that Nessett [NESS90] gives to demonstrate the flaw in BAN Logic. Nessett's protocol is given in Figure 2.1. Nessett's protocol is able to pass BAN Logic evaluation because the first message is encrypted under a key (pka^{-1}) that is known only by principal A. The problem is that the decryption key (pka) is well known. These conventions are unique to public key cryptgraphy which we will discuss in greater detail later.

Syverson defines decryption semantically to mean that a principal A can decrypt a

message if and only if both the encrypted message and its decryption key reside in every world possible for A. He explains that because in Nessett's protocol, both the message containing the potential session key (Kab) and its decryption key (pka) are common knowledge in all worlds, the message containing Kab may be decrypted by any principal. As a result, key Kab is not a secure key and, thus, not a good key.

Nessett's Insecure Protocol

A->B: $\{Na,Kab\}^{pka-1}$

B->A: $\{Nb\}^{Kab}$

Figure 2.1

2.1.1 The Logic of Authentication

As we mentioned earlier, Burrows, Abadi, and Needham introduced BAN Logic in [BAN88], [BAN90]. BAN Logic is a straightforward mechanism that allows reasoning about beliefs that principals may have during a protocol run. BAN Logic achieved widespread acceptance as the standard logic for protocol verification, though it has not achieved widespread implementation.

Burrows et.al. form BAN Logic around constructors for believing, seeing, controlling, and saying messages. The authors choose not to utilize functional notation, e.g. believes(P,X)) but opted for a more natural structure: P believes X. They do offer a predicate for freshness and other notation to represent the concepts of having a good public and private key and for characterizing secrets and formulas.

2.1.1.1. BAN Logic Constructs

While BAN Logic substitutes natural structure for predicate format, they introduce non-standard notation to represent key concepts, e.g. $<-^{k}->$ represents the fact that k is a good key. We present the BAN Logic constructs using predicate calculus notation in lieu of the actual BAN notation. Our representation of the BAN Logic constructs is given

below. The first three constructs represent actions on messages taken by principals during the protocol run. The remaining constructs reflect possible beliefs that the principal may have based on the contents of messages seen in a protocol run.

$\{X\}_k$: Message X is encrypted under key k. The only way someone that possesses this value can expose X is by decrypting the value using key k.

$[X]_y$: Message X is combined with password y. The presence of y proves the identity of the principal that generated the message $[X]_y$

A->B:(X): Principal A sends message X to principal B.

believes (A,X): When this construct appears in a proof, principal A has sufficient evidence to believe that statement X is true and will be true throughout this run of the protocol.

sees(A,X): This construct indicates that some principal (maybe A) sent message X and A received X.

said(A,X): Principal A sent message X.

controls(A,X): A has jurisdiction over X or A is an authority with respect to X. Effectively, if A has jurisdiction X and B believes that A believes X, then B can believe X.

fresh(X): X was generated in this run of the protocol.

goodkey(A,kab,B): kab has all the characteristics of a symmetric cryptographic key, suitable for securing communications between parties that share kab.

pubkey(A,ka): ka has all the characteristics of an asymmetric cryptographic key, suitable for securing communications between parties that share ka.

secret(P,X,Q): X is a secret between P and Q

$<X>_y$: X combined with formula Y. Y is a secret that proves the identity of the originator of $<X>_y$.

2.1.1.2. BAN Logic Rules of Inference.

The constructs above give us a logical language to use to describe principal's beliefs during a protocol run. The BAN Logic rules of inference provide the reasoning power to deduce new beliefs from existing beliefs.

The Message Meaning Rules. These three rules provide the authenticity-detecting power of BAN. The first two rules define how symmetric and asymmetric key cryptography prove the identity of a message originator. Simply, if I see a message that was encrypted under a certain key, then only the principal that possesses that key (or it's inverse for asymmetric key cryptography) could have sent the message. The third rule provides the same definition for shared secrets that are not keys, but that can be securely combined with some message.

For private keys:
believes(P,goodkey(P,k,Q)) and sees(P,{X}k) => believes(P,said(Q,X))

For public keys:
believes(P,pubkey(Q,k)) and sees(P,$\{X\}k^{-1}$) => believes(P,said(Q,X))

For shared secrets
believes(secret(Q,y,P)) and sees(P,$<X>_y$) => believes(P,said(Q,X))

Nonce-verification rule. While the message meaning rules allow principals to reason about what other principals said, the nonce verification rule allows principals to reason about what other principals believe. Loosely, if another principal said it, and if it is fresh, then the principal that said it, believes it.

believes(P,fresh(X)) and believes(P,said(Q,X)) => believes(P,believes(Q,X))

Jurisdiction rule. The jurisdiction rule extends the principal's reasoning ability even further. This rule allows a principal to gain a belief based on what another principal believes.

believes(P ,controls(Q,X))and believes(P,believes(Q,X)) => believes(P,X)

Sees rules. The Sees rules define how principals gather information during a protocol run. The first rule indicates that catenated messages are easily decoupled and the second reflects the ability of any principal to extract a message from its signature. The last three sees rules define the encryption and, by default, the decryption operations. Asymmetric key encryption requires two rules to define the reciprocal nature of the two keys.

sees(P,<X,Y>) => sees(P,X)
sees(P,[X]y) => sees(P,X)
believes(P,pubkey(P,kp)) and sees(P,{X}kp) => sees(P,X)
believes(P,goodkey(P,k,Q)) and sees(P,{X}k) => sees(P,X)
believes(P,pubkey(Q,k)) and sees(P,{X}k-1) => sees(P,X)

Freshness rule.

believes(P,fresh(X)) => believes(P,fresh(<X,Y>);

The BAN Logic rules of inference establish many, sometimes subtle, assumptions. The Message Meaning Rule, for example, assumes that the cryptosystem utilizes perfect encryption and the nonce verification rule assumes that a principal believed a message when they "said" it. Yet, it is in these eleven rules separated into five categories that BAN Logic gains its verification power. These rules describe the relationships between seeing, saying, controlling, and believing messages, formally define the impact of a message being fresh, and define the meanings of the encryption and decryption operations.

2.1.1.3 BAN Logic Analysis of the Needham and Schroeder Private Key Protocol

In order to illustrate BAN Logic analysis, we now analyze the Needham and Schroeder Private Key Protocol [NS78] in Figure 2.2. The analysis is largely taken from [BAN90].

The first nine statements of the proof are assumptions given in BAN Logic notation. All are intuitive and non-contentious, with the exception of statement 9, which we will address more directly in a moment. Statements (10), (17), (22), and (27) are annotations. These are derived by the protocol evaluator based on the protocol statements. The protocol steps themselves are idealized using the guidance: "Roughly, a real message m can be interpreted as a formula X if whenever the recipient gets m he may deduce that the sender must have believed X when he sent m" [BAN88,p330]. This idealization is not as straightforward as deriving the assumptions and annotations were.

Because the guidance for idealization is so vague, and because different evaluators may have different perceptions regarding protocols and their environment, it is easy to envision different evaluators constructing different idealizations of the same protocol. In the Needham and Schroeder example, the second statement is idealized with the "fresh(kab)" predicate. The protocol evaluator was able to recognize that the authentication server was trying to convey this notion because of the convention the protocol specifier utilized in naming the nonce. Passing fundamental information such as this through

naming conventions can easily result in misunderstanding or contradiction. For example, some protocols utilize timestamps, named "tx" or certificates named "cx" as conventions to establish the freshness of messages. Evaluators unfamiliar with all such conventions would not be able to generate accurate idealizations for such protocols.

BAN Logic Analysis of the Needham and Schroeder Protocol

1 - A and S believe goodkey(A,kas,S)
2 - B and S believe goodkey(A,kbs,S)
3 - S believes goodkey(A,kab,B)
4 - A and B believe S controls kab
5 - A believes S controls fresh(kab)
6 - A believes fresh(na)
7 - B believes fresh(nb)
8 - S believes fresh(kab)
9 - B believes fresh(kab)
A->S: A,B,na
S->A: {na,B,goodkey(A,kab,B),fresh(kab),{goodkey(A,kab,B)kbs}kas
10 - A sees {na,B,goodkey(A,kab,B),fresh(kab),{goodkey(A,kab,B)}kbs}kas (A)
11 - A believes S said
(na,B,goodkey(A,kab,B),fresh(kab),{goodkey(A,kab,B)}kbs) (MM,1,10)
12 - A believes fresh (B,goodkey(A,kab,B),fresh(kab),{goodkey(A,kab,B)}kbs) (F,6,11)
13 - A believes S believes fresh(kab) (NV,11,12)
14 - A believes S believes goodkey(A,kab,B) (NV,11,12)
15 - A believes fresh(kab) (J,5,13)
16 - A believes goodkey(A,kab,B) (J,4,14)
A->B: {goodkey(A,kab,B),A}kbs
17 - B sees {goodkey(A,kab,B),A}kbs (A)
18 - B believes S said (goodkey(A,kab,B),A) (MM,2,17)
19 - B believes fresh(goodkey(A,kab,B)) (F,9,18)
20 - B believes S believes (goodkey(A,kab,B)) (NV,18,19)
21 - B believes goodkey(A,kab,B) (J,4,20)
B->A: {nb,goodkey(A,kab,B)}kab
22 - A sees {nb,goodkey(A,kab,B)}kab (A)
23 - A sees (nb,goodkey(A,kab,B)) (S,16,22)
24 - A believes B said (nb,goodkey(A,kab,B)) (MM,16,22)
25 - A believes fresh(nb,boodkey(A,kab,B)) (F,15,24)
26 - A believes B believes goodkey(A,kab,B) (NV,24,25)
A->B: {nb,goodkey(A,kab,B)}kab
27 - B sees {nb,goodkey(A,kab,B)}kab (A)
28 - B sees (nb,goodkey(A,kab,B)) (S,18,27)
29 - B believes A said (nb,goodkey(A,kab,B)) (MM,18,27)
30 - B believes fresh(goodkey(A,kab,B)) (F,7,28)
31 - B believes A believes goodkey(A,kab,B) (NV,29,30)

A - Annotation
MM - Message Meaning rule
NV - Nonce Verification rule
J - Jurisdiction rule
F - Freshness rule
S - Sees rules

Figure 2.2

Said another way, idealization requires the protocol evaluator to have global understanding of the messages and message components in the proof in order to accurately idealize the protocol. For example, the evaluator must understand exactly the meaning a nonce will have for the recipient and how it will be used in order to correctly idealize that nonce. We believe this global intent must be included in the protocol specification by the protocol originator to reduce ambiguity and to free the evaluator to focus more on determining if the protocol meets its goals, and less on determining how the protocol is supposed to meet them.

In our example, we do not list the actual protocol steps as statements in the proof, but rather show where they occur sequentially to give form to the proof. While this may give the impression that the order has some effect on the proof, it in fact does not. The only sequencing involved in this proof is deductive sequencing. For example, justification of statement 20 relies on statement 19. While statement twenty must have been derived after statement 19, this has nothing to do with the order of execution of the steps. Nonetheless, we show the steps sequentially in order to facilitate understanding of the proof.

The remainder of the statements are derivations generated using the BAN Logic rules of inference given above. The derivations are justified in parenthesis following the statements. The desired results of the proof are found in statements 26 and 31. As Burrows, et.al. point out, these results are obtained by making the "unusual" assumption found in statement 9 that B assumes the new key passed from the authentication server is fresh. As was pointed out in [DS81], this unusual assumption is flawed, but necessary because there is no evidence of freshness sufficient to derive this as a theorem after the third protocol step, between statements 17 and 21.

2.1.1.4 Summary of BAN Logic Analysis.

BAN's strengths lie in the simplicity of its logical language and the small number

of constructs and rules of inference that it requires. The primary weakness of BAN Logic is the fact that it is not complete; that is, while evaluation with BAN Logic can detect some errors, protocols with flaws can avoid detection by BAN Logic [NESS90].

BAN Logic, like many of the other proposed logical systems, has no mechanism for enforcing strict ordering on the steps in the protocol. The BAN Logic meaning of each statement depends only upon the annotation rule which allows a principal to "see" a message when it is sent to them, so the BAN Logic meaning of any statement in a protocol is the same regardless of the statements before (or after) it. We see this as a weakness in the BAN Logic idealization process, which fails to retain the semantics added to the protocol by the sequence of the steps in the procedural definition. Snekkenes [SNEK91] shows that the loss of this meaning creates a dilemma for BAN Logic, in that "step permutable" protocols with obvious flaws may pass BAN Logic evaluation.

This weakness in BAN Logic is a result of the nature of the BAN Logic method. BAN Logic is used to prove properties about a specific set of protocol steps executed in a specific sequence. The proof reflects the beliefs of valid principals in the protocol run. The weakness surfaces because BAN Logic cannot predict how a powerful, active intruder may reorder the sequence of the steps in a protocol run. We discuss static and dynamic evaluation of protocols in more detail in section 2.5.

In [BAN90b], Burrows et. al. acknowledge the fact that BAN Logic is not an all encompassing method for ensuring protocol security and state that the problem of providing a mechanism that can detect all errors in a protocol is "quite difficult". This suggests that in protocol verification, like program testing, we should not seek a perfect solution, but should attempt to find ways to gain confidence in the security of protocols.

2.1.2 Non-monotonic Logical Systems

Another shared characteristic of many existing protocol verification logical sys-

tems is that they are monotonic; that is, knowledge and beliefs cannot be changed or refuted during analysis. In [MOS89] Moser suggests that analysis of cryptographic protocols requires us to be able to continue reasoning in the presence of new evidence that effectively refutes previously held beliefs/knowledge. She contends that complete knowledge is rarely available and always expensive and that it is important to be able to distinguish between information that is known with certainty and information that is based on trust in the behavior of participants. In practice, we depend on belief and trust rather than knowledge and enforcement. Since beliefs may be refuted, it is necessary to be able to continue reasoning if an assumption is refuted.

In support of her argument, Moser proposes a non-monotonic logic, i.e. a logic that allows reasoning to continue in the face of refutation of existing beliefs. Moser's nonmonotonicity is accomplished by combining a logic of belief with a non-monitonic "unless" operator. The unless operator establishes a relationship between two beliefs for a given principal. Annotated "$B_i(p)$ unless $B_i(q)$", the operator means that principal i believes proposition p unless i believes proposition q. Clearly, this notion adds complexity to the logic. For example, it may be possible to have more than one solution to a given formula. Moser gives the example that "$B_i(p)$ unless $B_i(q)$ and $B_i(q)$ unless $B_i(p)$" has two possible solutions, "$B_i(p)$ and $\sim B_i(q)$" or "$\sim B_i(p)$ and $B_i(q)$". As Moser states, "... rational agents may interpret the same evidence but reach different conclusions that are equally valid."

Introduction of the concept of refutation of beliefs requires some method of deciding which value of a refuted belief is to be utilized. If principal i believes proposition p to be false, but then discovers evidence that p is true, how would i decide whether to believe that p is true or false. Intuition suggests that we should accept the most recently acquired belief, under the assumption that some action subsequent to acquiring our original belief changed the state of the environment. Without temporal operators, we cannot resolve the

conflict in this way. Rather, Moser elects to resolve all potential paradoxes and contradictions in the direction of the absence of belief. Using this methodology, the formula "~B_i(p) and B_i(p)" simplifies to ~B_i(p).

In order to fully understand her handling of contradictory beliefs, consider the underlying logic. While some logical systems assume non-belief in the absence of evidence for belief, Moser's logic assumes belief in the absence of refutation. Effectively, if a principal i sees proposition p, then B_i(p). Thus, belief in any proposition p may be more accurately represented as "B_i(p) unless B_i(~p)". Additional information confirming the truth of p has no impact upon the formula. However, once evidence is provided indicting the falsity of p, p becomes false in the formula and cannot be "reconfirmed" as true. Propositions are considered to be universally true or false. If p is false, it will never become true and vice versa.

In [SYV91] Syverson argues that the non-monotonic features of reasoning about cryptographic protocols should be accomplished outside the logical system because of the "tremendous increase in difficulty of reasoning..." that are required to incorporate these features into the logic. He suggests that providing the unless operator greatly complicates reasoning, while it provides little or no additional flexibility. The utility of changing assumptions may be accomplished as easily by returning to the original argument list and restarting the reasoning process.

2.1.3 Temporally enhanced logical systems.

The discussion of non-monitonic logical systems touched on the issue of temporal considerations of protocols. These issues arose because of the recognition of two types of attacks active intruders may use against protocols. The first attack, known as a replay attack [DENN81], involves having an intruder record messages from a protocol session and use the copied messages to trick a participant in a current session. The discussion of

whether or not a message originated in the current protocol session is considered to be a matter of freshness. A message is considered to be fresh if it is not older than some time threshold, usually corresponding to the start of the protocol session.

The second temporal issue is of the order of messages (or possibly components of messages) in the current session. It was shown that unless protocol messages are selected very carefully, active intruders may be able to extract parts of previous messages in the current run that will trick a participant into an invalid belief [BIRD92].

Neither of these issues is directly concerned with time, but rather they are a matter of sequencing. One method of dealing with sequencing is to use time to distinguish which message did, or should have, occurred first. These issues lead to development of temporally enhanced logical systems containing operators to deal with freshness and with the sequencing problems of BAN Logic identified by Snekkenes [SNEK91]. The methods described below are examples of these logical systems.

2.1.3.1 A Logic of Knowledge and Time

In [BIEB90], Bieber describes his Logic of Knowledge, Time and Communication, CKT5. The foundation for CKT5 is a simple language based on the AND, OR, and NOT connectors and a set of propositional variables. While there are no specific temporal operators, the knowledge operators are indexed by the responsible agent and the time of the action, e.g. principal A knows predicate F at time t.

Bieber adds to the logic of knowledge and time the communications modal operators SEND and RECEIVE. He defines the SEND operator to mean that once a principal A sends a message, then A knows that some other principal receives it. Conversely, the RECEIVE operator means that if B receives message m, then m is a usable message and B knows that some principal sent m. To be usable, a message must be either a valid cleartext message, or a valid cleartext message encrypted under some valid key one or more times.

With the connections in place between knowledge, time, and communication, Bieber introduces the encrypt and decrypt functions to allow interpretation of messages. Categories of messages include clear, clean, computable, catenated, elementary, partially clear, and univoque. By combining the constructs and the long set of accompanying definitions, the logic allows one to deduce security related information about the protocol.

Bieber's contribution is in defining the send, receive, encrypt and decrypt operators in such a way that successful reasoning about cryptographic protocols executed in a hostile environment may be accomplished. The send and receive operators provide the link between communication and knowledge. The encrypt and decrypt operators (termed functions by Bieber) quantify the logic and provide the means to protect the interpretation of messages against anyone that does not possess the appropriate key, for if two agents share a private key, the inability to decrypt messages guarantees secrecy and the ability to encrypt messages guarantees authentication.

Bieber's reasoning builds from the definition of send and receive, which mean that if principal A sends or receives a message m, then A and one other principal knows m. The interpretation of m is derived from the concepts of a cleartext message, message encryption [or decryption], and message catenation. If a message received by A is constructed using only cleartext messages, catenation, and encryption/decryption with a key known by A, the message is "computable" and thus can be known by A.

In his conclusion the author notes, "The major difference between their approach [Burrows, Abadi, and Needham] and ours lays on the difference between belief and knowledge." Bieber calls his a "realistic" view of communication because the "usable" definition of knowledge does not infer any trust into other agent's behavior.

2.1.3.2. GNY.

In their logic (called GNY) [GNY90], Gong, Needham and Yahalom extended and

enhanced BAN Logic. The temporal extension of GNY, termed "P is told a formula that he did not convey previously in the current run" reflects the author's attempt to provide a measure of sequencing of protocol steps. If a "not originated here" message is received by a principal A, and if A can determine that the message is fresh, then A did not convey the message in this run of the protocol. While this operator narrows in on the problem, it does not provide a mechanism to detect if an intruder is replaying a message that was sent by a different principal (perhaps one that knows the shared key) in this or a previous run.

Two other BAN Logic extensions of GNY deal with distinguishing between a principal recognizing or possessing and believing a message. Rather than utilizing a general belief operator, GNY requires context for the belief operator. A principal can believe that statements are true, public keys are valid, messages are fresh, another principal conveyed a message, or that secrets are well-kept. In a sense, these operators provide type checking for the belief operator.

It is also instructive to recognize the difference between the belief about public keys and private keys with GNY. The private key constructor indicates only that a proposed key k is a suitable secret between two principals. This is a more narrow representation of the characteristics of k than required in BAN Logic where the key is understood to have all the necessary characteristics of a good key. While these characteristics are not listed in the BAN papers (or elsewhere that we know of) secrecy is certainly one of the important characteristics. Others include unpredictability, no duplication, and, of course, the necessary encryption and decryption results. The public key belief of GNY reflects the same understanding of the public key as with BAN Logic.

2.1.3.3. Kailar and Gligor

Another example of a logic that attempts to deal with the step sequencing problem of logical evaluation of cryptographic protocols is given by Kailar and Gligor [KG91].

The system they call Ordered Knowledge Sets is an extension to BAN Logic. They provide a syntactic addition to BAN Logic that forces specification of the originator, recipient, and message round indicator with the message contents to formally order the protocol steps. In the evaluation, this information is used to construct ordered lists depicting when each principal believes principals gained knowledge of each data item they see.

Interestingly, it is the syntactic addition of Kailar and Gligor's verification mechanism that forces the specification of additional information used to do the verification. In [AN94] Abadi and Needham give principles for improving security of cryptographic protocols. In that paper, they recommend including information within each message that identifies the intended message round and protocol session.

2.2 - Use of specification and verification tools developed for software evaluation.

In [KEM89] Kemmerer describes a mechanism for cryptographic verification based on specification and verification tools developed for software evaluation. The idea of Kemmerer's approach is to formally specify the components of the cryptographic network and the associated cryptographic protocol actions. The Formal Development Methodology (FDM) state machine approach to formal specification is used. States are differentiated from one another by the values of state variables, which are changed only by well defined state transitions. Thus, the components are represented as state constants and variables, the protocol rules are represented as state transitions, and assumptions about the cryptographic algorithms are specified as axioms. Because of the latter, one can verify the system assuming the use of a different encryption scheme by replacing the current axioms with axioms that express the properties of the different encryption scheme. The desirable properties that the protocol is to preserve are expressed as state invariants. The theorems that must be proved to guarantee that the system satisfies the invariants are automatically generated by the verification system.

The formal specification language that is used, Ina Jo, is a non-procedural assertion language that is an extension of first-order predicate calculus. Assertions are used to specify the critical requirements for a secure state. The system being specified can change state only as described by one of the state transforms. After the formal specification is completed, one can verify the theorems that are generated to check if the critical requirements are satisfied. If the theorems are verified and the encryption algorithms satisfy the assumed axioms, then the system will satisfy its critical requirements.

To begin the test, the user submits the formal specification to the Ina Jo testing tool Inatest, which allows the user to interactively monitor and control the test. A sample of the Inatest commands taken from [KMM93] is given in Figure 2.3.

FDM employs an inductive approach to generate the necessary proof obligations to assure that the critical requirements are preserved. One must show that the criteria hold in the initial state and for every transform, show that if the transform fires in a state where the criteria hold, then the resultant state also satisfies the criteria and that the previous and new states satisfy the relationships expressed by the constraints. That is, the initial state is the basis case and the induction is on the transforms.

Kemmerer utilizes his system to analyze a protocol given by Tatebayeshi, Matsuzaki, and Newman [TMN91] in [KMM93]. The critical requirement that the system is to satisfy in all states (called the criterion) for the TMN protocol is:

> "... the only key that is known to the intruder... and that is also a key that was used by the system is the Server's public key."

Figure 2.4, taken from [KMM93], gives the Ina Jo transforms for the Tatebayashi, Matsuzaki, and Newman (TMN) Protocol. The statements in the second column of Figure 2.4 are Ina Jo representations of the SN notation of the protocol steps in the first column.

Sample Inatest Commands

Add	- add a predicate
Exec [trans]	- execute transform 'trans'
FIle [fileid]	- read commands from the named file
Help	- display available commands
LS	- list specification
LT	- list transforms
Path	- display current path
Quit	- return to UNIX
Restore [N]	- restore state number 'N'
Save [comment]	- save a state
SEQ [fileid]	- execute a sequence of transforms
STATES	- display saved states
Vars [id]	- display [one] variable value(s)
Check [current or Result]	
Display [current or Start or Result]	
Init [start or Result]	

Figure 2.3

An advantage of expressing the system using formal notation and attempting to prove properties about the specification is that if the generated theorems cannot be proved, the failed proofs often point to weaknesses in the system or to an incompleteness in the specification. These may take the form of additional assumptions required about the encryption algorithm, weaknesses in the protocols, or missing constraints in the specification.

One important characteristic of Kemmerer's system is that by including the intruder as part of the formal specification, it can be used to detect attacks by active intruders as well as evaluating static protocols against passive attacks. Kemmerer suggests that there are three approaches to analyzing encryption protocols:

1) Formal verification of the protocol specification

2) Exhaustively test the protocol
3) Intelligently select particular scenario's for testing

Tatebayashi, Matsuzaki, and Newman Protocol in SN and Ina Jo

SN	Ina Jo Transform Declaration
A->S: (A,B,e{ka}pks)	Request_Key(A,B:User, ka:Key)
S->B: (A)	Process_Request(A,S:User, A:Text)
B->S: (e{kab}pks)	Respond_To_Server(A,B:User,kab:Key)
S->A: (e{kab}ka)	Return_Key(A,S:User, C:Text)
A->B: (e{important_data}kab);	Get_Key(A,B:User,important_data:Text)

Figure 2.4

Kemmerer's system uses both 1 and 3. The system does not search through scenarios to find vulnerabilities, but rather leaves selection of a scenario up to the protocol specifier. Kemmerer's system then tests the formal specification of the scenario against attack. If the analyst does not test the appropriate scenario, no flaws will be revealed. In [KEM89], Kemmerer showed how his method was used to reveal a previously known flaw in a protocol, though the method has not successfully detected a previously unknown flaw in a protocol.

2.3 Testing tools based on expert systems which evaluate different scenarios.

The dominant technique for determining whether or not computer programs meet their goals is testing. Program testing has received broad and deep research in its ability to accurately determine if programs meet their goals. While testing is recognized as an incomplete verification mechanism, many testing methodologies exist that improve the testing process for computer programs. Millen et. al. use this approach for application to the CPV problem.

In [MCF87] Millen, Clark and Freedman offer a testing tool designed to be used for verification of the subset of cryptographic protocols called Key Distribution Protocols. Their tool is called Interrogator. Interrogator is a series of Prolog programs which take a black box view of the Key Distribution System.

The input to Interrogator is a protocol specification and a data item. The Interrogator utilizes the specification to build a state transition machine based on the transmit and receive operations. Once the machine with its transitions are built, Interrogator systematically exercises the paths of the state machine using its model of protocols and network communication. The goal is to determine if the intruder can learn private information. If a path is found that divulges private information to an intruder, the message history that allows the compromise is displayed.

The Interrogator utilizes exhaustive search for cryptographic protocol verification. On the positive side, if an attack exists, Interrogator will find it. However, for complex protocols, the size of the problem can become quickly intractable and the success of the process becomes dependent on the wiles of the analyzer for providing helpful hints for reducing the size of the search. The system is also limited by its selection of only key distribution protocols for evaluation.

2.4 Term rewriting systems

The NRL Protocol Analyzer [MEAD91] is an interactive Prolog program which was successfully used to identify flaws in existing protocols [MEAD92]. The tool is based on the algebraic term-rewriting method proposed by Dolev and Yao [DY83]. The NRL Protocol Analyzer makes use of protocol specifications as communicating state machines. Analysis is performed by showing that undesirable states are unreachable. The state of the machine is represented by the beliefs of the participants. Transitions are events in which words are generated and beliefs are modified. Transition rules have three

parts: pre-conditions, post-conditions, and event statements. Event statements are used to record the firing of a rule and indicate what the rule does. Specifically, it identifies the principal, the protocol round, the event, and the value of the principal's counter.

The NRL Protocol Analyzer utilizes a narrowing algorithm to find the substitutions necessary to determine if a state can be reached. To prevent an explosion of states from application of the algorithm, the search space must be limited. This can be done by the user with the help of several tools provided with the Analyzer.

2.5 Static Versus Dynamic Protocol Evaluation

There are two perspectives that must be considered when evaluating the effectiveness of a protocol. The characteristics of the evaluation methods differ significantly and have been the subject of much debate in the literature. The first view we should consider is what the protocol accomplishes with regards to the trusted principals. This perspective corresponds to static evaluation of a protocol, i.e. evaluation of the meaning of the protocol given that its steps are executed in the exact sequence and with the exact content that was intended by the designer.

In considering the results of proper actions of trusted principals, it is reasonable for us to view and evaluate the actions and accumulated beliefs (belief sets) of both principals during the protocol run, since we are assuming that each principal is doing what they are supposed to be doing. Accordingly, we may desire to check the belief set of principals against each other to determine if they correspond. In order to be able to evaluate protocols from this view, any useful evaluation mechanism should provide a mechanism to explicitly bind each action in the protocol to the principal that is responsible for that action. The question of which party created a nonce, for example, can have a major impact on whether or not the nonce accomplishes its objective of ensuring message freshness. Without this feature, it could also be difficult to determine who has jurisdiction over

data items or who accomplished the encryption of a data item in a protocol.

The second perspective we must consider is how an active intruder might be able to compromise the protocol. We may categorize this as dynamic evaluation of a protocol, or more generally, the security of the protocol. In this category we consider how a powerful intruder may cause protocol steps to be executed out of sequence, or may convolute the content of messages from what the protocol designer intended. The intruder may accomplish this, for example, by recording messages from a previous session and replaying them into a current session (replay attack) or by initiating more than one session simultaneously and ingeniously tricking the trusted participants into erroneously mixing messages from different sessions (parallel session attack).

To illustrate the problem of active attacks, consider the following benign protocol. We assume that principals A and B share a common key k before the start of the protocol run.

A->B: (A);
B->A: ($\{N\}^k$);
A->B: ($\{N+1\}^k$);

If the protocol is executed as written, A can believe that the session is begun with B because N can be successfully decrypted with key k. Since only B knows k, B must have sent N. B can believe that a session is started with A because N+1 is successfully decrypted with key k. If an active intruder X records the above session, then waits for A to originate another session, X may be able to defeat the protocol by intercepting A's message intended for B and replying as follows:

A->B: (A); -- Intercepted by X
X->A: ($\{N\}^k$); -- Message replayed from the previous session
A->B: ($\{N+1\}^k$); -- Intercepted by X

Based on the above interaction, A would erroneously believe that a secure session is established with B.

Dynamic evaluation of protocols is more complicated than static evaluation. From this view, it is only reasonable to consider the beliefs of one principal at a time, since the principal cannot assume that they are communicating with a trusted principal. This evaluation must be made based exclusively upon what the principal sends and receives on the net. To accomplish this, each user must be able to remember things that have happened by storing and later retrieving messages, nonces, and keys as if they were operating in a private address space. Otherwise, it will be impossible to effectively reason about what a user knows or believes based on the information gathered via a protocol. In this scenario, it is not reasonable to assume what the contents of another principal's address space are, which further weakens our reasoning ability.

The significance of the problem of dynamic evaluation of cryptographic protocols is emphasized in research into the nature of attacks on cryptographic protocols such as in [AN94], [BIRD92], [BIRD93], [MOORE88], [SYV93a], [SYV93b]. By detailing the simple attack given above and other various attacks, these works highlight the variety and subtle nature of active attacks. Bird, Gopal, et al. and Abadi and Needham extend this research to offer guidelines for creating protocols which are not vulnerable to several of the known attacks. While there is no guarantee that the guidelines they offer are effective against as yet unknown attacks, this line of research appears to be helpful in finding a mechanism suitable for effecting CPV.

Chapter 3

The Cryptographic Protocol Analysis Language (CPAL)

As we described previously, the first step in our protocol evaluation process is the development of a language suitable for describing the actions of principals in a cryptographic protocol. CPAL contains the constructs necessary to express the protocols in a form similar to several of the ad hoc pseudocodes that are seen in the literature. Earlier, we discussed some features of these pseudocodes and utilized them to give the Needham and Schroeder and the Otway and Reese protocols in Figures 1.1 and 1.2. Carlson considered these pseudocode notations in [CARL94] naming one version Standard Notation (SN), and suggested that the syntax for SN is well defined, but that no semantics is associated with it. Figures 1.1 and 1.2 utilize the pseudocode form of SN. Throughout this dissertation we consider SN as the representative version of all pseudocodes invented for describing cryptographic protocols. We now look at these pseudocodes in some greater detail.

3.1 Pseudocodes for Cryptographic Protocol Specification

Because the pseudocodes prevalent in the literature evolved to describe the actions a principal may take in a protocol run, we are concerned with exactly what operations the protocol describes. Clearly the predominant actions principals take are to send and receive messages. As we pointed out earlier, the symbol -> in SN represents the send operation. There is no corresponding symbol for receiving a message. Message receipt is assumed within the send operation. Though rigor suffers for this omission, in most instances the notation is sufficient to portray the intent of the specifier.

The lack of a receive operation also complicates the meaning of names. Messages and message components are named as if they reside in some global address space. Names are retained throughout the protocol session as the messages are passed back and forth, principal to principal. It is impossible to tell where a message originated or if a new message is generated

with the same name as a previous message.

The second important action principals take in a protocol run is to encrypt and decrypt messages. In SN, only encryption is represented. The assumption is made that if a principal possesses an encrypted message and the appropriate key, they will attain the message contents. Occasionally, an encrypted message is forwarded to a principal that does not possess the key necessary for decryption with the intent that the message will be forwarded to a principal with the appropriate key in a later protocol step. This may lead to confusion regarding exactly where a message originated and who actually possesses the content of the encrypted message.

S->A: $\{na,B,kab,\{kab,A\}^{kbs}\}^{kas}$ (3.1)

In statement 3.1 notice that the message contains the encrypted value $\{kab,A\}^{kbs}$. Without considering the entire protocol, it may not be clear whether $\{kab,A\}^{kbs}$ is encrypted by S, or sent to S by some other principal earlier in the protocol. In this case, a naming convention resolves the ambiguity. Because the convention of using the letter k suffixed by the lower case letter for the principals that share the private key, a message encrypted under kbs could only have been originated by principal B or principal S. Since the message containing $\{kab,A\}^{kbs}$ that B received originated from S, then B assumes that only S could have done the encryption. We know now that this entire line of reasoning ignores the fact that the encrypted message may have been forwarded any number of times, and may have originated from B itself.

Other conventions evolved with the intent of reducing confusion of pseudocodes. Alice and Bob, the "first couple of cryptography" [DIFF88], [GORD84], also known as A and B respectively, represent the canonical communicating principals. Alice is routinely the session originator and Bob the responder to the prompt. When a centralized authentication server is utilized in a protocol, it is routinely named AS or S, though occasionally KDC (for key distribution center). When an intruder is present, he is often given the identity I.

While SN accurately represents a good summarization of the pseudocodes found in the lit-

erature, many variations appear. Some prefer a picture portrayal of the principals as remote entities, such as in statement 3.2. Some go as far as showing each principal as a node with all messages emanating between the circles which represent the nodes in the graph. Some pseudocode languages make the round number (sequence number of a message within a protocol) a part of the protocol specification, while others add mechanisms to specify a session identifier. Other less noticeable variations include subscripting or superscripting all or parts of the encryption and decryption key and using function notation for encryption, decryption, send and receive operations

```
          {na,B,kab,{kab,A}kbs}kas
S  ------------------------------------------> A                    (3.2)
```

Pseudocodes were introduced to add formality to the process of describing cryptographic protocols, or the rules principals were to use to establish secure communication using cryptography. These interactions are intuitively seen as algorithmic, i.e. a step-by-step process.

Pseudocode languages are best known for their extensive use by computer programmers. The languages were needed because the programmers were not able to easily plan the necessary computations and data manipulations in the machine languages of the day. Pseudocode languages represent a language understandable by an abstract machine that operates in an environment more similar to the application setup than the actual implementation machine. Assembler languages and then compiled languages were steps in the process of allowing programmers to "think" in the syntax of the programming language.

The process is similar with cryptographic protocols, with the focus on moving data rather than performing computations. Sending a message may be thought of as copying data from one address to another. Because cryptographic protocols routinely involve only two communicating principals and the steps are executed sequentially, advanced control structures required in computer programs are not necessary.

In cryptographic protocols, there are three fundamental sets of actions:

1 - Send and receive messages
2 - Catenate messages and extract messages components from catenation
3 - Encrypt and decrypt messages

In SN, these actions are coupled into a single operator.

3.2 Protocols as Action Lists

It is common practice in the literature to specify protocols as execution traces of actions by two or more principals. The actions are normally intermixed, with the principals taking turns, alternately sending messages to one another. The protocols are expressed as though the actions are executed serially when in fact they are intended for execution by separate, cooperating processes, usually on different processors in remote locations.

A more accurate view of a protocol is as separate lists representing the actions of each principal, interleaved together to form an execution trace. The action lists contain all the actions each principal takes in a run of the protocol in the same order given in the protocol specification. The original protocol can be reconstructed from its action lists by interleaving the action lists to form what we term an execution trace. In reconstructing the execution trace, the original order of the action lists is maintained, and the separate actions are re-synchronized in the execution trace by the blocking nature of the send and receive operations.

As we describe in greater detail in the next section, send and receive may be thought of as atomic operations on a message queue. Principals use the send statement to place a message on the input queue of an intended recipient. The send is a blocking operation because if the queue of the intended recipient is full, the send operation cannot be completed. The receive statement, is used by a principal to retrieve a value from its own input queue. The receive statement is a blocking operation in that a principal cannot complete a receive operation until some principal (maybe the same one) executes a send operation with a message of the expected form and addressed to

that principal, i.e., until there is a message on the principal's input queue. Said another way, legal interleavings of action lists are formed by maintaining the order of action lists and synchronizing the send and receive operations of the principals involved.

We elected to represent the input queue of each principal as only one message deep. We do this because of the nature of most cryptographic protocols, where principals routinely send a message containing information and possibly a challenge and then wait [block] until a message reply is received. Steps are sequenced so that when a message is sent to a principal, that principal executes the next protocol action. We made these characteristics inherent in our system, rejecting protocols that violate the normal send and block protocol step sequence.

Needham and Schroeder Private Key Protocol Action Lists

A	S	B
A->S: A,B,na	S->A: $\{na,B,kab,\{kab,A\}^{kbs}\}^{kas}$	B->A: $\{nb\}^{kab}$
	A->B: $\{kab,A\}^{kbs}$	
	A->B: $\{nb-1\}^{kab}$	

Figure 3.1

SN is not suited for representing protocols as independent action lists. Figure 3.1 shows the Needham and Schroeder Private Key Protocol [NS78] action lists given in SN. One difficulty is the lack of a receive statement. In Figure 3.1, each action list begins with a send statement, so it is not clear which principal begins the sequence. Because the operations are indivisible, there can be no sequencing mechanisms in SN, such as blocking operations, available to serialize the steps.

3.3. The Meaning of CPAL Actions.

In discussing the meaning of actions in a protocol, we found it helpful to consider an infor-

mal model of the environment. For this purpose, we selected the model described in [AT91], where every message sent by a valid principal is intercepted by a powerful intruder. Upon intercepting a message, the intruder may select any of several courses of action, of which the interesting ones include:

- Forward the message to its intended destination as received,
- Save the message for later transmission,
- Modify the message and forward it as addressed,
- Combine parts of the message with other previous messages,
- Readdress all or parts of the message,
- Replace the message with its own or a previously recorded message.

For our purposes, there are three critical characteristics to the model. First, each principal operates in an independent address space, not visible or directly accessible to any other principal (including intruders). In this address space, the principal records the contents of the messages they have sent and received in this session. By recording the traffic from the network and analyzing the contents, the principal can decide what they believe. We represent the data items stored in the user's address space by using dot notation, with the field identifier prefixed by the owning prinicpal’s identifier. Hence we can represent key "kab" in A's address space as A.kab. This information can easily be gathered by a syntax checker and later utilized by a verification condition generator.

Second, the only way for data to flow from one address space to another is through messages sent and received during the protocol session. While intruders can passively listen to all network traffic, or can intercept, modify and reinsert messages, or remove messages from the network without detection, the principal's address space is considered to be private and secure from inspection or modification by anyone other than the owner. Furthermore, the only mechanisms available to the owner for adding data to the address space are via messages received from the network, generation of random values, and reasoning about data already in the address space.

Finally, each principal has an input queue through which they receive all messages. Sending of messages is modelled as the assignment of a message to the queue of the recipient. Receipt of a message is modelled as assignment from the principal's input queue into the same principal's address space.

The send operator can be thought of as a reliable send, i.e. the message that is sent will always end up on the input queue of the desired recipient. Possible intruder actions are modelled by sending messages to the intruder. In order to represent an all-seeing intruder, all messages may be sent to the intruder. In order to represent the benign or passive intruder, the intruder will merely forward messages as they were received to the desired recipient. In this way, the actions of the intruder are formalized, clearly specified, and are fully controlled by the protocol evaluator.

A more detailed description of our informal model is contained in Appendix A.

3.4. The Syntax of CPAL.

We now turn to the design features of the language we developed to resolve the difficulties with pseudocode notations. CPAL is a procedural language with a complete formal syntax. A Backus/Naur Form description of CPAL is provided as Appendix B. As we suggested above, we were guided in our design of CPAL by several goals, most based on resolving the pseudocode difficulties described there. Our first concern was that the language must be expressive enough to represent all significant actions of a principal in a protocol. This is why we model CPAL on existing pseudocodes used in the literature for specification of cryptographic protocols. Our second concern was that protocols written in a pseudocode, such as SN, could be easily translated into CPAL. We accomplish this by basing our CPAL statements on the principal’s possible actions in cryptographic protocols.

A CPAL protocol specification consists of a sequence of actions by principals. A CPAL action contains one or more CPAL statements. An action consists of the statements prefixed by the identifier of the principal performing the action. Each statement is followed by the statement

separator, the semicolon. If more that one statement is included in an action, the statements must be enclosed within curly braces ({}).

CPAL statements closely resemble the operations provided in pseudocodes like SN. As in SN, principals may send messages, generate random values, catenate values, and encrypt and decrypt values. In CPAL, principals may also receive messages, compare values, make decisions which change the flow of the protocol, and make assumptions and assertions regarding the environment and affects of the protocol. While several of these operations are not explicit in SN, they are present, either coupled in other operators or required to be included in a prose description of the protocol. The valid statement types in CPAL are listed in Table 1 within Chapter 4 of this paper.

For our purposes, a message is considered to be a value or binary bit pattern which can be represented on a computer. Values in CPAL are integers, strings, booleans, and compound values. Catenated values are designated by their inclusion in angle brackets (<>). Values may be changed by applying a function to them in the normal way.

While encryption and decryption operations closely mirror function calls, because of their essential part in cryptographic protocols, we give them special operators in the language. The most important characteristic of these operators is that the original encrypted message can only be recovered by decryption under the same key.

In CPAL, an encrypted value is enclosed in square brackets, prefixed by the operator "e" and suffixed with the key, which is also a value. A decrypted value is of the same format with the encryption operator replaced by the decryption operator, "d". The catenated value containing the encryption of the value x and the decryption of the value y under the key k is represented as:

<e[x]k,d[y]k>.

3.4.1. The Assignment Statement

Messages are non-destructively assigned in CPAL in the usual way, e.g., the source value is copied into the memory location of the destination identifier. As syntactic sugar, we allow omission of the principal name in assignment statements in the protocol since the only address space effected is that of the principal taking the action. The statement:

A: X := Y;

means the value of the address referenced by name X in A's address is replaced by the value stored in the addressed by the name Y in A's address space, or A.X := A.Y.

As we have mentioned, it is common practice in protocols to utilize newly generated values called nonces to ensure the freshness of messages. A principal may generate a nonce in CPAL by assignment of the value "new" to a variable, e.g.,

A: Na := new;

In cryptographic protocol runs, values are frequently catenated to form a new value. We represent a catenated value in CPAL using a structure we call a compound value, represented by X in the following assignment statement:

A: X := <W,Y,Z>;

For catenated messages to be useful, we must be able to "break out" the multiple identifiers contained in a compound value. In order to represent the components within compound values, we designed a "DOT" value. The numerical suffix to a dot value designation designates a member of the catenated list of values within the compound value. The expression

<W,Y,Z>.2

may be reduced to Y, while

X.3

represents the third value in the compound value X, in this example, Z. We may perform the same

operations on compound and dot values that are appropriate on any other value, such as transmission, encryption, decryption, and variable replacement. In fact, the value X.2 gains meaning when the identifier X is replaced with a catenated list, allowing reduction to the value indexed by the dot value suffix (in this case the second element in the list). For example if the identifier X were replaced by the catenated value <W,Y,Z>, the entire value structure <W,Y,Z>.2 could be replaced by the value Y.

As a convenience for working with catenated and compound values, the assignment

A: (D,E) := X;

means the same as

A: D := X.1;

A: E := X.2;

3.4.2. The IF Statement.

While decision operators are conspicuously missing from the pseudocode languages, CPAL provides operators based on the classic notion of a change in the flow of control of the protocol steps. If the given condition is true, a different statement is executed than if the condition is false. This is implemented using the IF structure where the condition is a boolean condition and the "then" and "else" subjects are alternative statements to be "executed". In the following example, if the condition Na==Na' is true at the execution of the action, A will encrypt the value msg under key kab and send the resulting value to principal B. If Na==Na' is false, A will encrypt the nonce Na and send the resulting value to AS.

A: if (Na==Na') then {=>B(e[msg]kab);}
else {=>AS (e[Na]kab);}

3.4.3. The SEND and RECEIVE Statements.

The dominant operation in pseudocode languages for cryptographic protocols is the send

statement. Again, the syntax we chose for send closely mirrors that of the pseudocodes. Statement 3.3 is the CPAL representation of statement 1.0 shown in Chapter 1 in pseudocode and repeated here for convenience:

A -> B: $\{X, Y\}^{kab}$ (1.0)

A: => B(e[<X, Y>]kab); (3.3)

The CPAL syntax for the send statement closely mirrors that of the SN syntax for send. Since the action identifier precedes each statement in CPAL, the originator's identifier is not repeated in the send statement itself. Thus, the send operator (=>) is always the first symbol in a send statement, followed by the identifier of the intended recipient of the transmission. The message body is enclosed in parenthesis and may contain any valid CPAL value as described in section 3.4.1. In statement 3.3 above, principal A is executing three operations: the catenate operation on messages X and Y, the encrypt operation on the catenated value, and the send operation on the encrypted catenation. Because this send operator results in the message going directly to the queue of the intended recipient, we also call this the secure send operator.

The receive statement in CPAL is structured similarly to the send statement. The principal that is to receive the message is identified as the acting identifier of the action, so the recipient's identifier is not repeated in the receive statement. By design, the receive statement may only receive one value, greatly simplifying the format of the receive statement, which has only two components: the receive operator (<-) and the name of the value to be received is enclosed in parenthesis.

As described in a previous section, we model an all-powerful intruder by sending messages from valid principals to the intruder for further transmission (or mischief). We introduce a second form of the send operator (->) which we call the insecure send operator, to facilitate this feature. Thus, if the analyst desires to reflect the reliable transmission of a message from the originator to the recipient, the secure send operation is in order. To reflect transmission through the

intruder, the insecure send operator is used. This enhances the ability of the analyst to model and formally define the meaning of the activity of an intruder, similarly to Kemmerer's intruder transformations discussed in section 2.2.

In the following statements, principal A composes the message "MSG", encrypts it under the key "kab", and sends the encrypted value to principal B. The intruder intercepts the message, then forwards it to principal B unchanged, representing the passive intruder in this instance. Principal B receives the encrypted value, decrypts it, assigns an identifier, and provides space in memory.

```
A: ->B(e[MSG]kab);
I:  <-(X);
I:  =>B(X);
B:  <-(X);
B: MSG := d[X]kab;
```

3.4.4. Expressing Assumptions and Goals

In order to allow expression of goals and assumptions in a protocol, we include operators for "asserting" and "assuming" the truth of a predicate. Expression of beliefs reflects the ability of a principal to accumulate and state knowledge or belief of received or derived facts, including the belief that the integrity of the protocol has been compromised.

Principals express beliefs in CPAL using the ASSERT statement to specify protocol goals to be proven and the ASSUME statement to specify assumptions to use in establishing the truth of the goals. Both goals and assumptions are stated as predicates, either as primitive predicate calculus symbols e.g. (X=Y and Q=R), or as a special predicate represented as a function of a list of parameters which is considered to yield a boolean value when evaluated, e.g. believes(A,X).

In the following example, principal A expresses the assumption of the equality of A's and

B's version of the session key between A and B. The system may use this assumption to prove or disprove the goals of the protocol. Later in the protocol, B expresses the goal that we desire to prove of believing that A sent MSG.

```
A: assume(A.kab == B.kab);
                    .
                    .
B: assert(believes(sent(A,MSG));
```

CPAL and SN Specification of a Trivial Protocol

	global: assume (A.k==B.k);
A->B: (A)	A: -> B(A);
	B: <- (A);
B->A: $\{N\}^k$	B: -> A(e[n]k);
	A: <- (msg);
	A: n := d[msg]k;
A->B: $\{N+1\}^k$	A: -> B(e[f(n)]k);
	B: <- (msg);
	B: n' := d[msg]k;
	B: assert (n' == f(n));

Figure 3.2

3.4.5. CPAL and SN

The simple protocol given in section 2.5 is shown Figure 3.2 in SN and in CPAL. Because we chose the CPAL constructs based on principal's actions, we can translate any protocol described in SN into a CPAL specification that encapsulates the SN meaning and allows enhanced ability to represent the actual meaning of the protocol. The following guidelines for translation are given:

1. For any SN send statement, generate a CPAL send statement with the same originator, recipient, and message contents. Also generate one receive statement with the action principal being the recipient of the message as designated in the send statement.

2. Receive statements receive only one value. If the value that is received is a compound value, encrypted value, etc., the value may be manipulated using assignment statements and the encryption and decryption operators as necessary to acquire needed message components.

3. Principals may originate messages by creating a new value, performing a valid operation on new or existing values, or through assignment of the "new" operator.

Goals and assumptions accompanying protocols specified in SN in prose may be entered into the CPAL protocol specification using assert and assume statements.

The comparative lengths of CPAL and pseudo-code representations of protocols are approximately five to one. When it is considered that CPAL allows encoding of assumptions and goals into the protocol while the pseudocodes usually do not, and that most protocols involve less than ten messages, this is an acceptable ratio.

Chapter 4
A Formal Semantics for CPAL

4.1. A Foundation for Semantics

One of the difficulties with methodologies similar to that of BAN Logic for conducting cryptographic protocol verification involves a difficulty in translating the procedural description (or definition) of the protocol into a logical definition of the protocol. The BAN Logic approach uses an informal process called *idealization* in order to accomplish this translation. We recognized the need to replace idealization with another, more formal method to effect this translation, and have adopted a notion taken from programming language verification to resolve this difficulty.

4.1.1 Precondition/Postcondition Reasoning

While the operational model, or informal semantics, for CPAL is helpful in assisting us in understanding what the operations do, it is not well suited to the formality we require to evaluate cryptographic protocols. To achieve the necessary level of formality, we selected a classic approach to program verification based on work by C.A.R. Hoare [HOAR69]. In this paper Hoare describes precondition/postcondition reasoning about programs and/or program segments. Hoare observes that we can define the meaning of a program statement or segment by defining what [pre] conditions must hold such that if that statement or segment is executed (and terminates), a certain [post] condition will hold after the execution. The notation used to describe that P is a precondition for segment S and postcondition R is:

P {S} R

For example, in order for execution of the specific program statement “y := sqrt(x)” to cause the postcondition"y == 2" to be true, the precondition "x == 4" must hold before the statement is executed. Thus,

x == 4 {y := sqrt(x)} y == 2

Once each operation in the language is given a precondition/postcondition definition, a segment may be evaluated for its semantics by establishing the desired postcondition and reasoning from the last statement to the first to determine the necessary segment precondition. In order to consider a simple example, we define the assignment statement as:

Q(x) {x := e} Q(e) Def 4.0

where Q is a predicate initially dependent upon x and e is any valid expression in the language. This says that for Q to be true after the assignment statement x := e is executed, the predicate Q with each instance of x replaced by e, must be true just before the assignment statement is executed.

In Table 1, we define statement catenation to be:

Q {S1;S2} P == Q {S1} R and R {S2} Q. Def 4.4

This says that in order for the segment S1;S2; to be correct with respect to precondition Q and postcondition P, the postcondition of S1 must be the precondition for S2. Consider the program segment:

1. x := y / z;
2. z := sqrt(x);

We select the goal of this segment to be z == 2. We now know to look for the necessary precondition Q such that:

Q {x := y/z; z := sqrt(x);} P

where P is (z == 2). By Definition 4.2, we must first find the condition R such that:

R {z := sqrt(x);} P

By definition 4.1, we know that R is equal to P with all instances of z substituted by sqrt(x). Thus, R is:

sqrt(x) == 2.

We can now look for Q such that:

Q {x := y/z;} sqrt(x) == 2

Once again we use definition 4.1 to find Q. For each instance of x, we substitute y/z to obtain:

sqrt(y/z) == 2 {x := y/z; z := sqrt(x);} z == 2

Since the sqrt operation is not a program statement, we do not need a precondefinition of sqrt in order to complete the proof. Rather, we need only apply well-known mathematical identities to Q to determine that our final definition is:

y/z == 4 {x := y/z; z := sqrt(x);} z == 2

Notice that y/z == 4 is just one precondition that satisfies the definition. The predicate:

((y==16) and (z==4))

also satisfies the equations. The difference is that the derived precondition (y/z==4) is more general, and in fact satisfies the precondition ((y==16) and (z==4)). Generality of preconditions was the focus of Dijkstra in [DIJK76] as we will see in the next section.

4.2 Weakest Preconditions

In [DIJK76] Dijkstra describes an extension of Hoare's precondition/postcondition mechanism called the weakest precondition for a program statement or segment. Dijkstra introduces the concept of a "predicate transformer' - "a rule telling us how to derive for any [arbitrary] post-condition R the corresponding weakest precondition for the initial state such that activation will lead to a properly terminating activity that leaves the system in a final state satisfying R." The weakest precondition Q for segment S and postcondition R is written:

Q = wp(S, R)

The precondition is considered to be weakest if its predicate transformer describes ***all*** states that enable the program segment to result with the desired outcome. By describing preconditions in their weakest form, the most general case is represented, meaning that the weakest precondition of a segment is actually a Verification Condition for the segment. That is, there are no additional preconditions for the segment. If the weakest precondition for that segment can be proven from the assumptions, the segment's goals are guaranteed to be accomplished upon execution.

To verify that a segment meets its goals using weakest preconditions, we work in reverse as we described above. The initial verification condition is the value TRUE and the segment goal is routinely the last statement in the protocol specification. When we apply the definition of the last statement, the verification condition becomes the weakest PREcondition for the last statement. That resulting verification condition is now the POSTcondition for the next to the last statement and so on. When the first statement is reached and its weakest precondition is evaluated, we have completed generating the verification condition/weakest precondition for the segment. To complete the verification, the verifier must show that the derived verification condition is implied by the assumptions of the start state.

4.3 Weakest Preconditions for CPAL Statements

The meaning of CPAL statements and segments are represented in terms of a predicate using the weakest precondition paradigm. CPAL statement and operator definitions are expressed as operations on the predicate, either modifying variables within the predicate or adding conjunctions and disjunctions to the predicate. We developed the definitions for CPAL statements similarly to those of programming constructs described in [DIJK76] and [WULF81].

In this section, we give the weakest precondition definition of CPAL statements. There are only twelve statement types in the CPAL language. Each statement type has a weakest precondition definition. Because assignment is the most common operation in cryptographic protocols and because the "substituting" nature of weakest precondition definitions is best illustrated by assignment, the weakest definition for the assignment statement is an important definition for CPAL.

4.3.1 Statement Catenation

The definition of the catenation of statements is critical to our selection of this model for formal semantics. It is this definition that allows us to force sequencing upon the steps in a cryptographic protocol during analysis. The definition we use is similar to that described in [DIJK76] and [WULF81]. For statements S1 and S2 and postcondition P, in order for predicate P to be true after "S1;S2;" is executed, the weakest precondition of S2 for P must be true after execution of [i.e., be a post condition for] S1 for P. This is formally stated in definition 4.4 and given here for convenience.

$$wp(S1; S2, P) = wp(S1, wp(S2, P))$$

We illustrate this concept with an example in Section 4.5.

4.3.2 The ASSIGNMENT Statement

In our operational model, we think of assignment in the ordinary way: a value is copied from location referenced by the source identifier to the location referenced by the destination identifier. Our formal definition is more abstract, capturing the "copying" nature of the assignment statement. As we mentioned in the earlier example, Definition 4.1 means that for Q to be true after the assignment statement x := e is executed, the predicate Q with each instance of x replaced by e, must be true just before the assignment statement is executed

In our operational model we consider each principal to have their own address

space. To effectively represent this concept, fully qualified variable names consist of the variable name prefixed with the name of the principal in whose address space the identifier exists. For example, the fully qualified name for value x existing in principal A's address space is A.x. When it is clear by the context which address space the value exists in, the prefix may be omitted.

X and e are values residing in the address space of principal A. "e" represents an expression while x is a named value. Q is a predicate potentially dependent on A.x (i.e. A.x is a named value in some formula in predicate Q). If Q is in fact not dependent on A.x, then the assignment statement has no affect on Q. However, if P is dependent on A.x, then in order for Q to be true after the assignment statement is made, Q would have also had to been true before the assignment statement with every instance of A.x replaced by A.e.

4.3.3 The SEND Statement.

Again referring to our operational model, we can think of the send operator as a reliable assignment to the message queue of the intended recipient. During evaluation, we use the convention of appending 'q' to the receiving principal's identifier to indicate we are addressing the queue of that principal. Intuitively this definition means that we replace all references to the recipients's queue in predicate P with the value "msg" from the originator's address space.

4.3.4 The RECEIVE Statement.

The receive statement represents assignment of the value in the receiver's message queue into the receiver's private memory. Thus, the send/receive cycle involves two steps:

1. Assignment of the message from the originator's address space to the recipient's queue

2. Assignment of the message from the recipient's queue to the recipient's address space.

Our model reflects the power of the intruder to intercept messages at will. In order to facilitate the representation of intruder actions, CPAL includes an insecure send that does not assign the message to the queue of the intended recipient, rather assigns the message to the queue of the intruder. For an insecure send through an intruder, there are at least four steps necessary in order for a principal to transfer a value to another principal's address space. First, the originating principal executes a send operation, placing the intended message on the intruder's input que. The intruder receives the message, placing the message into its address space. At its discretion, the intruder may then execute a send operation, copying the message onto the input queue of the recipient. Finally, the recipient receives the message into its address space.

Table 1: CPAL WEAKEST PRECONDITION DEFINITIONS

Operation	Weakest Precondition Definition		#
Assign	wp("A: x:= e;", P(A.x)) =	P(A.e)	4.0
Comp Asgn	wp("A:(X,Y,Z):=W;",P(A.X,A.Y,A.Z))=	P(W.1,W.2,W.3)	4.1
Send	wp("A: ->B(msg);", P(Iq.msg)) =	P(A.msg)	4.2a
Secure Send	wp("A:=>B(msg);",Q(Bq.msg)) =	Q(A.msg)	4.2b
Receive	wp("B: <-(msg);", Q(B.msg)) =	Q(Bq.msg)	4.2c
Ifthenelse	wp("if C {S1} else {S2;}", P) =	C -> wp("S1;",P)) & (~C -> wp("S2;",P))	4.3a
Ifnoelse	wp("if C S1;", P) =	C -> wp(S1,P)) & (~C -> wp(skip,P))	4.3b
Stmt Cat	wp("S1; S2", P) =	wp(S1, wp("S2;", P))	4.4
Assert	wp("assert(R):",P) =	(P and R)	4.5a
Gassert	wp("gassert(R):",P) =	(P and R)	4.5b
Sym enc	wp("d[e[X]k1]k2",P(d[e[X]k1]k2) =	(k1==k2) -> P(X) and (k1 != k2)->P(new)	4.6a
Asym enc	wp("dp[ep[X]k1]k2",P(dp[ep[X]k1]k2)=	(-k1==k2)->P(X) and (-k1!=k2)->P(new)	4.6b
Assume	wp("assume(R):",P) =	(not P or R)	4.7

Here we encounter a difficulty in passing the names of principals in a protocol. Occasionally, it may be necessary to assign a variable name to a received value represent-

ing the name of a principal that will be a destination of a message. This situation results in having multiple names for a principal in a protocol run and complicates the send/ receive weakest preconditions. We illustrate the issue in the following code:

```
A: =>S(<A,B>);
S: <-(src,dst);
S: =>src(e[<dst,newkey]kas);
A: <-(msg);
```

In the second and third statements of the segment above, principal S uses an alias for principal A (i.e. src). Receipt of the value into the variable name src is routine. It is when principal S attempts to send a message to principal whose name is represented by src that the difficulty arises. Recall that the SEND definition is implemented by replacing the appropriate value in the weakest precondition predicate which represents the placement of the transmitted value on a queue. When SEND is executed in this instance, the verification condition generator must be able to determine which queue should receive the value, since src is a variable.

We handle this condition in the same way that Dijkstra [DIJK76] defines assignment to an element of an array, i.e. to mean that S will send the message to the principal whose name is aliased at the time of the send. Formally,

```
wp("A:=>X(msg);",P(Bq.msg)) =
  (((X==A)->P(A.msg)) and
  (X!=A)->P(msg)) and
  ((X==B)->P(B.msg))  and
  (X!=B)->P(msg)) ...  for all principals in the protocol.
```

The problem may be avoided by thoughtful selection of names and by exercising care in determining who the recipient of each message really should be. For this reason,

and because of the additional space and computation required in this definition, we do not implement this definition in the current version of our system.

4.3.5 The IF THEN ELSE Statement

As we saw with the definition for statement catenation, since weakest preconditions are themselves predicates, weakest preconditions may be recursively defined; that is, the weakest precondition for one segment may be dependent upon, or defined in terms of weakest preconditions of other segments. The weakest precondition definition of the IF statement (Table 1, definition 4.3a) is also defined in this way. For a predicate P to be guaranteed to be true after the IF statement

if C then {S1;}else {S2;}

is executed, truth of the condition C must imply the truth of the weakest precondition of statement S1 with respect to P. This is because when the if statement is executed, the truth of C guarantees execution of S1. The analogous situation with S2 holds if C is false. That is, the falsity of C must imply the truth of the weakest precondition of statement S2 with respect to P. The meaning of the IF statement without the else (Table 1, definition 4.3b) is similarly defined if we accept the meaning of the statement "skip;" to be: "take no action".

4.3.6 The ASSERT statement.

The ASSERT statement differs from the previously given statements in that it does not represent any action by a principal in a protocol run. Rather, the ASSERT statement provides the protocol specifier a mechanism to insert intended goals in the appropriate place in the protocol steps. By issuing the ASSERT statement

A: assert("kab is a good key");

the protocol specifier is stating that principal A can logically believe that "kab is a good key" is true at this point in the protocol. This predicate represents a goal to be proven by

the verification technique. Since no action is taken by this statement, in order for an arbitrary predicate P to be true after this assert statement is executed, it must have also been true before the statement was executed. Hence, definition 4.5a applies. Without going into the details of the underlying logic, the result of the application of the ASSERT statement definition is to add a conjunctive condition to the predicate defined thus far.

The GASSERT statement is identical to the assert statement except that the acting ID suffix must be included with each identifier. This allows the comparison of values across address spaces.

4.3.7 The Assume Statement

Like the ASSERT statement, the ASSUME statement does not involve an action on any address space. The ASSUME statement enables us to modify the predicate that is being defined based on logical analysis. We know that assumptions may be used to prove theorems. If Q is the theorem we are attempting to prove and P is an assumption, we can state:

P |- Q,

or equivalently,

P IMPLIES Q,

which is logically equivalent to:

Q OR NOT P.

This is the Weakest Precondition definition for ASSUME as we see in definition 4.7.

While the ASSERT statement only adds conditions to be proved to the predicate, the ASSUME statement also gives us “truths” that may be removed from the predicate by

replacing them with the boolean truth value TRUE. In implementation, because changes to the predicate caused by one assumed value may create an opportunity for additional "reductions" from other assumptions, the assumed values are applied repetitively until no further changes to the predicate occur. The assumed predicates are also output to a file to allow easy interface with other systems that may treat assumptions differently.

4.4 Encrypt and Decrypt operators

The definitions of the CPAL ***statements*** do not capture the entire essence of the CPAL language. Encryption and decryption, for example, are not statements at all, but are represented as operators on messages and message components. These operators are retained as part of a value in the predicate when the weakest precondition definition of the statement containing the encrypt or decrypt operator is applied.

The meaning of encryption and decryption are gained by surveying the predicate to find a suitable mixture of message components. In particular, when an encryption operation is found immediately within a decryption operation and if the keys of the operations are the same, the semantics of the encrypt and decrypt operators may be applied.

Intuitively, when symmetric key encryption is performed on a value X, it should mean that X cannot be retrieved unless the decrypt operator is applied to the encryption of X using the same key that was used to encrypt X. We implement this semantic concept with the rule given in definition 4.6a which says that in predicate P, if we find a decrypt/encrypt pair in which the keys k1 and k2 are equal, we can replace the entire decrypt/encrypt expression with the value X. If k1 and k2 are not equal, the value X cannot be derived from the expression, so we can reasonably replace the decrypt/encrypt expression with a unique value, a.k.a. a "new" value.

4.5 The "NEW" operator

The meaning of the "new" operator is also implemented as an operation on the predicate. Many cryptographic protocols rely on generation of nonces to guarantee the freshness of messages or to bind a response to another principal. The fundamental characteristic of these nonces is that every appearance of the nonce in the execution trace must be directly traceable to the point where the nonce was created. The semantic concept for the "new" operator is implemented as a replacement operation. Values of type "new" in the predicate are replaced with unique values generated by suffixing a base name with an integer counter. By doing this, each "new" value is guaranteed to have a unique name, and any duplicates must be considered to be copies of the original.

4.6 A Simple CPAL Verification Example

Figure 4.1 contains a simple protocol exchange between the principals A and S constructed to illustrate application of several CPAL weakest precondition definitions. In the first column, we give the protocol steps. In the second column, we show the step-by-step transformations on the weakest precondition as the definition for each statement is applied.

Because the weakest precondition definitions are applied in reverse order, the initial predicate state of TRUE is in the second column of line 8. When the definition of the last protocol statement is applied in line 7, the additional predicate to be proven provided in the ASSERT statement is added to the weakest precondition to be proved by the system. When the RECEIVE definition is applied in line 6, the notion of the value moving from the recipient's queue into their address space is reflected in the changing of the value A.Na' to Aq.Na' in the weakest precondition predicate. The remaining transformations are similar predicate substitutions.

The final predicate shown in line 1 is the verification condition for the protocol. If

that predicate can be proven from the assumptions, then the protocol is guaranteed to meet the goal it presented in line 7. In this case, the only assumption we need is that of the identity principle for values, which we may use to reduce the equivalence equation on the right side to TRUE. Thus, our simple protocol meets its specified goal. Often, proof of the final verification condition is much more difficult, as we shall see in the next chapter.

4.7 The CPAL Verification Condition

What we accomplished in the previous example was to establish a logical definition for the protocol based on, and directly corresponding to, the ***procedural definition*** of the protocol. The primary function of CPAL-ES is to translate the procedural protocol description given in CPAL into a predicate whose ultimate value is either true or false, just as in BAN Logic-type approaches. However, the meaning of the value of this predicate is different, and more formal than in BAN Logic.

CPAL-ES Evaluation Example

Protocol Statement in CPAL	Weakest Precondition Predicate Transformations
1. A: Na := new;	TRUE and ((A.unique.001-1)==(A.unique.001-1))
2. A: =>S(Na);	TRUE and ((A.Na-1) == A.Na-1)
3. S: <-(Na);	TRUE and ((A.Na-1) == Sq.Na-1)
4. S: Na' := Na-1;	TRUE and ((A.Na-1) == S.Na-1)
5. S: =>A(Na');	TRUE and ((A.Na-1) == S.Na')
6. A: <-(Na');	TRUE and ((A.Na-1) == Aq.Na')
7. A: assert((Na-1)==Na’);	TRUE and ((A.Na-1) == A.Na')
	TRUE

Figure 4.1

By allowing the declaration of the protocol goals within the CPAL description as we saw in statement 7 of Figure 4.1, the meaning of the procedural definition of the protocol description is fundamentally different from the SN representations that BAN Logic-type approaches were designed to evaluate. Protocol specifications without goals included are merely a series of actions by principals. The intent of the actions must be inferred by the content of the messages or superimposed later based on a prose description by the protocol specifier. CPAL specifications, on the other hand, provide not only the capability for the protocol designer (or analyst) to include the goals of the protocol in the procedural specification, but give a formal meaning to those goals that can be automatically assessed by CPAL. In fact, formal specification of the protocol goals is essential to the CPAL evaluation process, as we will see in the next chapter.

CPAL evaluation is further distinguished from BAN Logic and other similar approaches, because it is formal. Once the specification is completed, derivation of its meaning is based only on the form of the specification. The essential elements of "form" described in this chapter are the CPAL statements, the CPAL operators, and statement catenation. It is the latter of these three mechanisms that allows our logical definition to reflect the information inherent in the sequencing of the protocol steps. Because of this sequencing power, any meaningful reordering of protocol steps will result in a different protocol meaning (i.e. a different verification condition).

The predicate that results from CPAL-ES evaluation reflects whether or not the protocol accomplishes the goals specified within the protocol steps. The complete predicate will always be a representation of the variables expressed in the goal of the protocol. Thus, if the goal is simple and the protocol is not complex, the final predicate may be as easily evaluated, as is the example in Figure 4.1. For more complex protocols and goals, we need automated assistance to assess the resulting predicate.

Chapter 5
The CPAL Protocol Evaluation System

5.1 CPAL As a System

The ultimate goal of our efforts is to establish a system that allows efficient assignment of a meaningful definition to a broad variety of cryptographic protocols, including sophisticated attack sequences on protocols. Toward that end, we developed the formal language CPAL, described in Chapter 3, and a formal semantics to accompany CPAL, given in Chapter 4. In this chapter we give the method we use to integrate these components, along with other less novel mechanisms into a practical tool to use for evaluating cryptographic protocols. We call this method the CPAL Evaluation System, or CPAL-ES.

5.1.1 Specifying Protocols in CPAL

The first step in the CPAL-ES process is to encode the target protocol in CPAL. If a Standard Notation (SN) representation of the protocol is available, the guidelines given in Chapter 3 for translating SN specifications into CPAL may be used. However, no SN representation is required and the protocol may be encoded directly into CPAL using the CPAL syntax description given in Chapter 3.

It is important to consider the nature of protocol goals and how goals for a protocol should be selected. BAN Logic-type evaluations seek to achieve goals based on the beliefs of principals, such as “A believes B believes kab is a good key”. These beliefs are established by combining the contents of the messages the principals see along with some understanding of the form of the protocol, and the characteristics of cryptography. As principals “see” more messages, they believe more things. These new beliefs may combine with other existing beliefs to form yet other new beliefs.

When the axioms of the logic are applied to the new belief set, it may lead to

another new belief, e.g. that a secure channel is established between A and B. Thus, with BAN Logic, we take the belief set of a principal as we assume it is, evaluate the protocol steps, then acquire new beliefs based on the rules of inference. If when no further beliefs can be acquired and BAN Logic is unable to acquire a "goal belief" specified external to the protocol steps such as the ones just stated, we assume the protocol is not secure and begin looking for problems with the protocol.

The goals that are the focus of the native CPAL-ES system may be considered to be message-based goals. CPAL-ES gives us a formal definition of message composition, message transmission, encryption, and decryption, and message possession. The goals that we evaluate are thus, based exclusively on the messages that are passed. Consider the goal given in the example in Figure 4.1, "Na-1 == Na'". This goal is based only on values passed in messages with no interpretation whatsoever. The value Na was created by A and the value Na' was received in a message. The stated equivalence can be irrefutably determined by a principal, requiring no interpretation. By including this relationship in an assert statement, we add this predicate to the list of facts CPAL-ES must prove about the protocol. Specifically, CPAL-ES must prove that after statement 6 of the protocol executes, the value Na in A's address space is one greater than the value Na' also in A's address space. The focus of CPAL is the specification of the actions of principals in a protocol. The success or failure of these actions are most often measured by the accomplishment of goals such as comparing versions of nonces to ensure the freshness of messages or checking encrypted values to ensure that keys match.

As we mentioned in Chapter 2, BAN Logic evaluation is based on what principals see, say, control, and believe. Beliefs are only changed by principals seeing and saying things. The content of what principals see and say is critical to this evaluation. Unfortunately, the goals of a protocol that BAN Logic and other logics depend upon for their meaning, are often met in subtle ways not intended by the protocol specifier. Once the

message based goals of the protocol are compromised, any reasoning based on the logics is not valid. In this way, CPAL-ES provides a clearer view of what a protocol means than Ban Logic-type approaches.

CPAL-ES allows the protocol verifier to try variations of the protocol to determine what other protocol action sequences meet the message-based goals. In a later section, we will discuss how message-based goals of a protocol may be met in ways other than that intended by the protocol creator. We will also see how CPAL-ES complements BAN Logic by allowing specification of desired beliefs as protocol goals.

As we see, selection of the proper protocol goals is essential to attaining a helpful definition of the protocol through CPAL-ES. Several rules of thumb apply to the selection of goals.

1. The last statement in a protocol should be a goal expressed in an ASSERT statement. Since CPAL-ES bases its evaluation on the values in the predicate, any statement other than ASSERT has no affect on the initial predicate. Thus, any CPAL statements after the last ASSERT statement are extraneous to CPAL-ES evaluation.

2. Use the GASSERT statement to evaluate matters of security. The ASSERT statement can only be used to consider comparison of data elements within the address space of one principal. Many important characteristics of cryptographic protocols rely on comparison of data elements within a single principal's address space. For example, principal A may send a message to B accompanied by a nonce. When A receives a reply assumed to be from B, A will compare the original nonce that was sent to the nonce that was received in the message from B to ensure the message is current. Other important goals may involve the comparison of values in different principal's address spaces or comparison of a value in a

valid principal's address space to one in the intruder's address space.

3. Special predicates may be used in ASSERT statements to allow evaluation of logical constructs other than the AND and OR constructs provided in CPAL-ES. Special predicates use function notation and allow evaluation of the special predicate parameters though the definition process. For example, we may express the desire to prove that A believes kab is a good key for principals A and B in the following statement:

```
GASSERT (A.believes(A.goodkey(A.A,A.kab,A.B)));
```

CPAL also allows the protocol designer to explicitly list intruder actions in the protocol specification. The principal name "I" is reserved in CPAL-ES for the intruder, and any message sent using the insecure send operator (->) goes to principal I. The intruder performs all other actions similarly to valid principals in the protocol. The intruder has its own address space and cannot access values in the private address spaces of other principals. We found that an intruder with these characteristics can effectively model a wide variety of passive and active attacks, many of which are given in Chapters 6 and 7.

5.1.2 Attaining the Verification Condition for a Protocol

Once the specification is complete, syntax errors must be removed. CPAL is a parsed language with a syntax checker that may be used interactively in either the UNIX or the DOS/WINDOWS environment to quickly scan a protocol specification for syntax errors. As a small language with few constructs and because of the short length of cryptographic protocols, correction of the syntax is accomplished easily and in a short period of time for most protocols.

Once the specification is free of syntax errors, the formal semantics of the proto-

col is automatically derived by CPAL-ES. This derivation is accomplished in two steps:

1. The definition of each statement is applied to an initial predicate with value TRUE, beginning with the last protocol statement and progressing in reverse to the first statement.
2. The resulting predicate is scanned for conditions allowing replacement reflecting the meaning of the encryption and decryption operators and value catenation and extraction. When such a condition is encountered, the definition for these operations is applied. Any designer-provided assumptions are also applied to the predicate where applicable.

5.1.3 Proving the Verification Condition

The final step in the CPAL-ES process is to attempt to prove that the verification is true given the assumptions with respect to the environment. The predicate resulting from the steps described in the last section may contain a repetition of logical conditions that may be easily reduced through routine simplification. Thus, CPAL-ES scans the predicate for conditions which may allow simple logical reductions. When these conditions are encountered, the appropriate value is reduced to its simplified form. A detailed description of the logical simplifications is provided in the next section. Because routine simplification may produce values that meet the conditions for other simplifications or for CPAL definitions that could not be successfully applied previously, this step is repeated and combined with the search for the encryption and catenation operation searches until all matches are found and the predicate is in its most simple form.

With the routine simplifications out of the way, the job of proving the verification returns to the protocol analyst. Occasionally, the simple reductions just described result in a final value of TRUE as was the case with the example in Figure 4.1. Frequently, the analyst will be left with a more challenging predicate to prove. Often, analysis of these com-

plex predicates by the analyst reveals the assumptions that are necessary in order to complete the proof. Consider, for example, the following verification condition:

(A.Na == d[e[A.Na]A.k]B.k)

We observe in this predicate that the right side of the equivalence statement fits the form of the decryption of the encryption of a value. If the keys of these operations were the same, the entire encryption/decryption value could be replaced by the value A.Na. Thus, if we insert the statement:

A: assume(A.k==B.k);

into the protocol, the final predicate will automatically reduce to TRUE. This assumption reflects the idea that principals A and B share a common key without giving any reference to how this key was distributed. We can also describe the preexistence of the shared key by giving a key exchange sequence, replacing the given assume statement at the beginning of the protocol. Because weakest precondition evaluation detects values that have the same origin, the statements:

A: =>B(k);
B: <-(k);

will allow proper decryption of values encrypted with A.k and B.k interchangeably. For the two equivalent versions of a private key, decryption inverts the previous encryption.

An analogous, situation occurs when public key encryption is used. Recall that with public (or asymmetric) key encryption, decryption using the encryption key will not invert the encryption. Rather, to invert public key encryption, the decryption function must use a key that is the inverse of the encryption key, or that “decrypts” the encryption key. To represent this situation, we introduce the special predicate:

global.decrypt(k1,k2)

This predicate proposes that the relationship between keys k1 and k2 is that public key decryption under k2 inverts encryption under k1. The predicate is positional so the predicate:

(dp[ep[A.X]k1]k2 == A.X)

reduces to TRUE given the assumption:

assume(global.decrypt(k1,k2))

The same is not the case if the position of the keys are exchanged as in the global.decrypt predicate:

assume(global.decrypt(k2,k1))

Whether or not such assumptions are valid for a particular protocol is a matter for the analyst to decide. Nonetheless, CPAL-ES will point the analyst to the missing or incorrectly stated assumption by giving the definition of the protocol exactly as it was stated.

5.2 Simplification of the Verification Condition

A shortcoming of weakest precondition for general program proofs is that verification conditions may be very long and complex. Fortunately, cryptographic protocols are substantially simpler than computer programs for two reasons. The first is that they are typically shorter, with most protocols being less than twenty CPAL statements long. The need to minimize the number of messages due to communications considerations ensures that protocols do not expand greatly in length. The second reason is that cryptographic protocols have no need for iteration or other complex flow control structures. Sequence and decision are the only flow control needed to satisfactorily express the actions of principals in a cryptographic protocol.

Even given the simplicity of protocols, the verification condition with several conditions, assertions, and encrypt/decrypt operations in some cases are up to a page in length. Fortunately, we have found that this length is greatly reduced by reduction of logical identities and elimination of message based assumed predicates.

5.2.1 Logical Identities

The logical identities we automatically reduce are tautologies in any logical system based on propositional calculus. These simple reductions allow us to reduce the verification condition for even intricate protocols to a simple predicate or even to a boolean value. The identities we implemented in the simplification portion of this system are given in Figure 5.1.

LOGICAL IDENTITIES SIMPLIFIED IN CPAL

P and P = P
P or P = P
P and ~P = FALSE
P or ~P = TRUE
P and TRUE = P
P or TRUE = TRUE
P -> Q = Q OR ~P
~(~P) = P
P == P = TRUE

FIGURE 5.1

5.2.2 Elimination of assumed predicates

Recall that assumptions are represented as predicates that are given to be true during the protocol run. These predicates are detected and eliminated from the verification condition in the simplification step. After the standard simplification is performed on the verification condition, the remaining predicate is searched for matches to any of the predicates specified in an ASSUME statement. Matches found to assumed predicates are

replaced with the value TRUE in the predicate and the simplification is repeated. For example, if the assume statement

A: assume (A.k == B.k);

exists in the protocol being evaluated and the verification condition simplifies to

(A.k==B.k) and (N1==N1'),

then simplification of the ASSUME statement will reduce the predicate to:

TRUE and (N1==N1').

The next iteration of the standard simplification will further reduce the predicate to:

(N1==N1').

5.2.3 Semantic reductions

As we discussed in Section 4.4, encryption and decryption semantics are applied to the predicate when an appropriate combination of decrypted and encrypted values occur together. We wait until the verification condition is completely constructed and all statement definitions are complete before scanning for such conditions. When a suitable combination is found, the substitution described in Section 4.4 is made to the predicate.

In Chapter 3, we describe how catenated values may be “extracted” from their catenated list. In implementation, we scan the predicate looking for an appropriate combination of dot values and catenated values. When the subject of a dot value in the predicated is identified as being a catenated value, the dot value is replaced by the member of the catenated list given by the dot integer.

5.2.4 Handling of More Complex Predicates

While our experience shows that many protocols produce verifications that can be easily simplified using the routine methods described above, it is conceivable that some protocols may not reduce so easily. We offer two methods to deal with these more complex verification conditions.

First, all final verification conditions are transformed into canonical normal form using the algorithm described in [CM81]. Standard proof techniques can be applied to prove the verification condition in this form.

Secondly, because of its ability to reflect assumptions and goals as arbitrary predicates, CPAL-ES may be used with automated theorem proving systems. The Boyer-Moore Theorem Prover [BM79], [BM88] and the Prototype Verification System [OSR93a], [OSR93b}, [SOR93a], [SOR93b] are two operating theorem proving systems. We address automatic theorem proving systems in detail in Chapter 7.

Chapter 6

Verifying Protocols Using The CPAL Evaluation System

The CPAL Evaluation System (CPAL-ES) is a three step process: the protocol is specified in CPAL, the verification condition is generated, and the attempt is made to prove the verification condition from assumptions about the environment. We now consider some specific examples of protocols verified using CPAL-ES. First, we will look at a simple, one-way authentication protocol and walk through the weakest precondition definition of that protocol. We then consider other protocols that illustrate the characteristics and features of CPAL-ES, progressively moving to more complex protocols with broader goals and different cryptographic environments. Finally, we illustrate how CPAL-ES reveals active attacks on protocols.

6.1 A Simple Cryptographic Protocol Verified.

In [BIRD92] the authors illustrate several attacks on protocols. The trivial one-way protocol they use, given in Figure 6.1, part (a), is convenient for illustrating output from CPAL-ES. In this simple protocol, the two principals use the knowledge of a common key to authenticate the communication session. The originating principal generates a random number, or nonce, to ensure the freshness of the interaction, encrypts the nonce under the common key, and forwards the ciphertext to the target principal, B. B receives the nonce, decrypts it, then sends the nonce back to the originator in the clear.

Since the originator believes that only the recipient could have decrypted the nonce and since the number was random and no intruder could have guessed it, the originator believes that the intended recipient decrypted the nonce and that the session is started with the desired principal. The authentication is one way because only the originator has assurances about the identity of the other participant. Since anyone can generate a

random number, the recipient has no information regarding the identity of the originator and no way to determine even whether the number is actually encrypted or not.

Figure 6.1, part a is constructed to show how the weakest precondition definition of a protocol is attained. The predicate listed to the right of each protocol step represents the weakest precondition after the weakest precondition transform for that statement has been applied. The process begins by applying the weakest precondition definition to the predicate TRUE.

We begin our manual evaluation in Figure 6.1, part a by applying the weakest precondition of the ASSERT statement given in statement 11. For simplicity, and because it has no impact on the final verification condition, we omit the TRUE branch of the predicate throughout this example.

For this protocol, the rest of the statements are substitutions of values on the predicate. Statement 10 shows principal A receiving N'1. This results in the value stored in A's address space (A.N'1) being replaced by the representation of that value on A's queue (Aq.N'1). The queue value is then replaced by the value in the intruder's address space in the secure send operation of the intruder in step 9 and so on. Note that we utilize the insecure send operator in this example only for illustration. In this case, the intruder acts as a passive listener, forwarding messages on without modification.

Steps 2 and 6 offer slight variations on the replacements in the other statements. In these two steps, a value represented by a single identifier is replaced by an encrypted or decrypted value. Recall that in our model, values may be complex structures including catenated values, “dot” values, functions and nested values. Regardless of the structure of the value, substitution in the predicate occurs in the same way.

A Trivial One-way Authentication Protocol

a

	Protocol action	Verification Condition at each step
1	A: A.N1 := new;	(unique.v0==d[e[unique.v0]A.k]B.k)
2	A: -> B(e[A.N1]A.k);	(A.N1 == d[e[A.N1]A.k]B.k)
3	I: <-(msg);	(A.N1 == d[Iq.msg]B.k)
4	I: =>B(msg);	(A.N1 == d[I.msg]B.k)
5	B: <-(B.msg);	(A.N1 == d[Bq.msg]B.k)
6	B: B.N1 := d[B.msg]B.k;	(A.N1 == d[B.msg]B.k)
7	B: -> A(B.N1);	(A.N1 == B.N1)
8	I: <-(msg1);	(A.N1 == Iq.msg1)
9	I: =>A(msg1);	(A.N1 == I.msg1)
10	A: <-(A.N'1);	(A.N1 == Aq.N'1)
11	A: assert(A.N1 == A.N'1);	TRUE and (A.N1 == A.N'1)

b

A.k == B.k

Figure 6.1

Also notice that the value A.N1 in the predicate does not change until the last definition application for protocol step 1. A.N1 was created in A's address space in the first protocol step, and not manipulated in A's address space at any other point in the protocol. The manipulations of values occur as values are changed in the protocol by assignment or by sending and receiving. The “unique” identifier in part a of Figure 6.1 is a result of the NEW operation performed by principal A. Because the two identifiers being compared are both instances of the same unique identifier, one must be a copy of the other. The verification condition suggests that if the decryption under key A.k of the encryption under B.k of unique.v0 returns unique.v0, then the protocol functions as desired.

The simplified predicate in part b of Figure 6.1 illustrates our definition of the encryption and decryption functions. If keys A.k and B.k are the same, the decryption operation reverses the encryption operation leaving the equivalence of the unique values as an identity that simplifies to TRUE. The outcome of TRUE indicates that execution of the protocol steps in the specified order and content results in the success of the message based goal given in step 11. Nonetheless, while the protocol meets the given message based goal, Bird et. al. show that this protocol is vulnerable to active attack by a sophisticated intruder. The attack they give on the protocol is similar to the attack we show later in this chapter on the two-way protocol of Bird et al.

6.2 Common Cryptographic Protocols Verified.

6.2.1 The Proposed ISO Protocol

While the example in Section 6.1 illustrates several concepts important to CPAL-ES, we will show the power of the system in this section by evaluating more realistic protocols. We begin with the proposed ISO protocol given in [BIRD93] and shown in Figure 6.2. In the manually generated weakest precondition shown in Figure 6.1, we showed the predicate at each step in the evaluation.

The evaluation in Figure 6.2 was taken directly from the CPAL-ES evaluation. As you see, CPAL-ES provides three components to the evaluation:

1. A listing of the protocol with the owner ID attached to each variable.

2. The verification condition is derived from the protocol statements before any simplification is performed

3. The simplified predicate, reflecting the final verification condition for the protocol

In many ways, the ISO protocol is similar to the protocol of Figure 6.1. No key is passed in the proposed ISO protocol and the equivalence of a received value to a previ-

ously transmitted original value is the message based goal of the protocol.

The unsimplified verification condition of the ISO protocol is larger and more complex than that of Figure 6.1 and gives some insight into just how difficult these predicates may be to evaluate. Nonetheless, our routine simplification steps quickly and easily reduce this predicate into a simple form. As with the trivial protocol, the simplified verification condition is the equality of the keys. This says that the protocol does not work if the keys are different, which is precisely the outcome we desire. In this case, it makes sense to assume the equivalence and the protocol meets its specified goals.

CPAL-ES Evaluation of the Proposed ISO Protocol

a

```
A: A.Na := new;
A: => B(A.Na);
B: <-(B.Na);
B: B.Nb := new;
B: => A(<e[B.Na]B.k,e[B.Nb]B.k>);
A: <-(A.msg);
A: (A.Nae,A.Nbe) := A.msg;
A: A.Na' := d[A.Nae]A.k;
A: A.Nb := d[A.Nbe]A.k;
A: assert(A.Na == A.Na');
A: => B(A.Nb);
B: <-(B.Nb');
B: assert(B.Nb == B.Nb');
 *** End of Protocol ***
```

b

```
((TRUE and (unique.v0==d[<e[unique.v1]B.k,e[unique.v0]B.k>.2]A.k)) and(unique.v1
== d[<e[unique.v1]B.k,e[unique.v0]B.k>.1]A.k))

****** End of Verification Condition ******
```

c

```
(B.k == A.k)
```

Figure 6.2

While the evaluation in Figure 6.2 shows that the protocol meets its stated goals, the interpretation of the meaning of the accomplishment of these goals is debatable. The

creators of this protocol may have believed that satisfaction of the given goals allowed each principal to believe they had established a secure communication session with the other, but Bird et. al. [BIRD93] showed how an active intruder can trick a principal using this protocol into believing a session is established with another valid principal when the session is actually with the intruder. We consider active attacks on cryptographic protocols in Section 6.4.

6.2.2 Woo and Lam

In [WL92], Woo and Lam offer a two-way authentication protocol similar to the ISO Protocol. The Woo and Lam protocol is a three party protocol that requires five transmissions to complete the authentication. No private key is assumed to be preexisting between the principals. The protocol is founded on two premises, one that a message received encrypted under a private key shared with another principal must have originated from that principal and, two, that it is impossible for even the most sophisticated intruder to guess the value of a nonce, so any message containing the nonce must be a component of a current protocol run. The CPAL-ES evaluation of the Woo and Lam protocol is given in Figure 6.3.

We see that in Figure 6.3, simplification of the original verification condition removes all references to variables, leaving the value of the final verification as TRUE. This would not be the result without the two assume statements that begin the protocol. Without these assumptions, the simplified verification condition would be:

(B.kbs == S.kbs) and (A.kas == S.kas)

which is similar to the final verification conditions of the first two examples we considered. The use of the assume statements allows us to make the final verification simpler and to explicitly state the conditions that comprise our assumptions.

Woo and Lam

```
S: =>A(kas);  -- The first 4 statements reflect prior shared keys
A:<-(kas);
S: =>B(kbs);
B: <-(kbs);
A: => B(A.A);
B: <-(B.A);
B: => A(B.Nb);
A: <-(A.Nb);
A: => B(e[A.Nb]A.kas);
B: <-(B.msg);
B: => S(e[<B.A,B.msg>]B.kbs);
S: <-(S.msg4);
S: (S.A,S.msg4b) := d[S.msg4]S.kbs;
S: S.Nb := d[S.msg4b]S.kas;
S: => B(e[S.Nb]S.kbs);
B: <-(B.msg5);
B: B.Nb' := d[B.msg5]B.kbs;
B: assert((B.Nb == B.Nb'));

 *** End of Protocol ***

(TRUE and (B.Nb ==
    d[e[d[d[e[<A.A,e[B.Nb]S.kas>]S.kbs]S.kbs.2]S.kas]S.kbs]S.kbs))

****** Simplified predicate follows.

TRUE
```

Figure 6.3

6.2.3 The Andrew Secure Remote Procedure Call Protocol

Another interesting protocol is the Andrew Secure Remote Procedure Call protocol given in [BAN90] and shown in Figure 6.4. The intent of this protocol is for two principals to utilize an existing shared, secret key in order to establish a new, shared, secret, session key. The fundamental differences between this protocol and the ISO protocol are that:

1. Nonces are not passed in the clear.

2. A function is applied to the nonce to give more assurance of the integrity of the protocol.

3. The principal receiving the request generates the session key.

4. The purpose of the protocol is to successfully pass a new session key from one principal to the other.

5. This protocol has three message based goals rather than two.

The verification condition for the Andrew Secure Remote Procedure Call protocol in Figure 6.4, part b is more complex than those from the previous examples. In this protocol, there are three comparisons with at least one value containing five levels of nesting, which illustrates that the length of the verification condition is a factor of the complexity, as well as the length, of the original protocol.

CPAL-ES Evaluation of the ANDREW Protocol

a

```
A: A.Na := new;
A: => B(<A.A,e[A.Na]A.kab>);
B: <-(B.msg);
B: (B.A,B.msg2) := B.msg;
B: B.Na := d[B.msg2]B.kab;
B: B.Nb := new;
B: => A(e[<B.f(B.Na),B.Nb>]B.kab);
A: <-(A.msg);
A: (A.Na',A.Nb) := d[A.msg]A.kab;
A: assert((A.f(A.Na) == A.Na'));
A: => B(e[A.f(A.Nb)]A.kab);
B: <-(B.msg3);
B: B.Nb' := d[B.msg3]B.kab;
B: assert((B.f(B.Nb) == B.Nb'));
B: => A(e[<B.k'ab,B.N'b>]B.kab);
A: <-(A.msg3);
A: (A.k'ab,A.Nb') := d[A.msg3]A.kab;
X: gassert((B.k'ab == A.k'ab));
 *** End of Protocol ***
```

b

```
 (((TRUE and (B.k'ab == d[e[<B.k'ab,B.N'b>]B.kab]A.kab.1)) and
(B.f(unique.v0)==d[e[A.f(d[e[<B.f(d[<A.A,e[unique.v1]A.kab>.2]B .kab),
unique.v0>]B.kab]A.kab.2)]A.kab]B.kab)) and (A.f(unique.v1) ==
d[e[<B.f(d[<A.A,e[unique.v1]A.kab>.2]B.kab),unique.v0>]B.kab]A.  kab.1))
**** End of verification condition ****
```

c

```
 ((not (A.kab == B.kab) or ((not (A.kab == B.kab) or ((not (A.kab == B.kab) or    ((not
(A.kab == B.kab) or (B.f(unique.v0) == A.f(unique.v0) and (A.f(unique.v1) ==
B.f(unique.v1))))  and (B.kab == A.kab))) and  ((A.kab == B.kab) or  ((not (A.kab ==
B.kab) or ((B.f(unique.v0) == unique.v3) and  (A.f(unique.v1) == B.f(unique.v1))))   and
(B.kab == A.kab))))) and  ((B.kab == A.kab) or  ((not (A.kab == B.kab) or ((not (A.kab
== B.kab) or ((B.f(unique.v0) == A.f(unique.v8))  and (A.f(unique.v1) == unique.v9)))
and (B.kab == A.kab))) and ((A.kab == B.kab) or    ((not (A.kab == B.kab) or
((B.f(unique.v0) == unique.v3) and (A.f(unique.v1) == unique.v13))) and (B.kab ==
A.kab)))))))and ((A.kab == B.kab) or ((not (A.kab == B.kab) or ((not (A.kab == B.kab) or
((not (A.kab == B.kab) or ((B.f(unique.v0) == A.f(unique.v0))   and   (A.f(unique.v1) ==
B.f(unique.v5)))) and (B.kab == A.kab))) and ((A.kab == B.kab) or ((not (A.kab ==
B.kab) or ((B.f(unique.v0) == unique.v3) and (A.f(unique.v1) == B.f(unique.v5))))   and
(B.kab == A.kab))))) and ((B.kab == A.kab) or  ((not (A.kab == B.kab) or ((not (A.kab ==
B.kab) or ((B.f(unique.v0) == A.f(unique.v18)) and (A.f(unique.v1) == unique.v19))) and
(B.kab == A.kab))) and ((A.kab == B.kab) or    ((not (A.kab == B.kab) or ((B.f(unique.v0)
== unique.v3) and (A.f(unique.v1) == unique.v23))) and (B.kab == A.kab))))))))
                **** End of simplified predicate ****
```

d

```
(B.f(unique.v0) == A.f(unique.v0)) and (A.f(unique.v1) == B.f(unique.v1))
```

Figure 6.4

It is also notable that in the evaluation of the Andrew RPC protocol, the simplified version of the verification condition, shown in part (c), appears to be much more complex than the unsimplified version in part (b). However, if we look carefully at the predicate in part (c), we see that many of the elements of the predicate are a repetition of the comparison of the originally assumed private shared keys of the principals, (A.k==B.k). If we acknowledge that the original keys are equal and add the statements:

A: =>B(k);

B: <-(k);

to the protocol, the verification condition reduces to the simplified predicate shown in part (d). From this final version of the verification condition we see that the ability of the protocol to meet its goals is dependent on the equivalence of the function that principals A and B apply to the nonces. Once again, it is fair to assume this equivalence and the protocol meets its stated goals.

6.3 Evaluating the Classic Protocols

6.3.1. The Needham and Schroeder private key protocol, given above in Figure 1, is often referenced in literature and is the foundation for the Kerberos [MNSS87] authentication system. The Needham and Schroeder Private Key Protocol is a three party protocol, that attempts two-way authentication. A significant contribution when it was introduced, it is now famous primarily for the potential flaw that it contains. As discussed previously, the flaw allows an active intruder to substitute a message from a previous run of the protocol into the current run of the protocol. If an intruder has been able to compromise a previous session key, the privacy and integrity of the session may be compromised without the originator detecting the compromise. BAN Logic evaluation of the Needham and Schroeder Private Key Protocol was given in Chapter 2.

We give the CPAL-ES evaluation of the Needham and Schroeder Private Key Protocol in Figure 6.5. Note the clear specification of the assumption of the shared keys between the authentication server (S) and principals A and B. The goals of the protocol are also identified in the two ASSERT statements in the protocol specification. As we discussed in the previous chapter, these goals represent the essence of the protocol. The only basis the principals have to establish beliefs in this protocol is the ability of the other principal to successfully decrypt a message using the shared private key. Comparison of the nonce stored by the principal to the value received in a message and decrypted with the private key provides the assurance the principals need.

The CPAL-ES evaluation indicates that if we assume the equivalence of the participant's private keys and that the two parties share the function f with the trusted server, then the stated goals of the protocol are met. Notice that even though CPAL-ES shows that the Needham and Schroeder protocol meets its message-based goals, there is a flaw in the protocol that makes it insecure. The difficulty arises because CPAL-ES verifies the protocol based on the explicit intruder actions stated in the protocol specification. By showing exactly what action is taking place and who is taking the action, CPAL-ES ensures that the stated goal will, or will not, be accomplished based on execution of the exact protocol steps specified.

CPAL-ES Evaluation of Needham and Schroeder Private Key Protocol

a

```
S: =>A(S.kas);
A: <-(A.kas);
S: =>B(S.kbs);
B: <-(B.kbs);
A: => S(e[<A.A,A.B,A.N1>]A.kas);
S: <-(S.msg);
S: (S.A,S.B,S.N1) := d[S.msg]S.kas;
S: => A(e[<S.N1,S.B,S.kab,e[<S.kab,S.A>]S.kbs>]S.kas);
A: <-(A.msg3);
A: A.temp2 := d[A.msg3]A.kas;
A: (A.N1a,A.B,A.kab,A.ticket) := A.temp2;
A: assert((A.N1 == A.N1a));
A: => B(A.ticket);
B: <-(B.ticket);
B: (B.kab,B.A) := d[B.ticket]B.kbs;
B: => A(e[B.N2]B.kab);
A: <-(A.msg4);
A: A.N2 := d[A.msg4]A.kab;
A: => B(e[A.f(A.N2)]A.kab);
B: <-(B.msg5);
B: B.N3 := d[B.msg5]B.kab;
B: assert((B.f(B.N2) == B.N3));
 *** End of Protocol ***
```

b

```
((TRUE and (B.f(B.N2) ==
d[e[A.f(d[e[B.N2]d[d[e[<d[e[<A.A,A.B,A.N1>]S.kas]S.kas.3,d[e[<A.A,A.B,A.N1>]S.ka
s]S.kas.2,S.kab,e[<S.kab,d[e[<A.A,A.B,A.N1>]S.kas]S.kas.1>]S.kbs>]S.kas]S.kas.4]S.k
bs.1]d[e[<d[e[<A.A,A.B,A.N1>]S.kas]S.kas.3,d[e[<A.A,A.B,A.N1>]S.kas]S.kas.2,S.kab
,e[<S.kab,d[e[<A.A,A.B,A.N1>]S.kas]S.kas.1>]S.kbs>]S.kas]S.kas.3)]d[e[<d[e[<A.A,A.
B,A.N1>]S.kas]S.kas.3,d[e[<A.A,A.B,A.N1>]S.kas]S.kas.2,S.kab,e[<S.kab,d[e[<A.A,A.
B,A.N1>]S.kas]S.kas.1>]S.kbs>]S.kas]S.kas.3]d[d[e[<d[e[<A.A,A.B,A.N1>]S.kas]S.kas.
3,d[e[<A.A,A.B,A.N1>]S.kas]S.kas.2,S.kab,e[<S.kab,d[e[<A.A,A.B,A.N1>]S.kas]S.kas.
1>]S.kbs>]S.kas]S.kas.4]S.kbs.1)) andc(A.N1 ==
d[e[<d[e[<A.A,A.B,A.N1>]S.kas]S.kas.3,d[e[<A.A,A.B,A.N1>]S.kas]S.kas.2,S.kab,e[<S
.kab,d[e[<A.A,A.B,A.N1>]S.kas]S.kas.1>]S.kbs>]S.kas]S.kas.1))
```

c

****** Simplified predicate follows.

(B.f(B.N2) == A.f(B.N2))

Figure 6.5

Denning and Sacco [DS81] devised an attack on the Needham and Schroeder protocol by creating an alternative trace that satisfies the message based goal of the protocol in a way the authors had not considered. Denning and Sacco's "replay attack" is now a well known attack and cryptographic protocols are routinely examined for vulnerability to replay. The replay attack is just one way that an intruder can manipulate a protocol run so that success of the message based goals does not guarantee the implications originally desired by the user.

As we stated earlier, like BAN Logic, CPAL-ES provides static protocol evaluation. If the steps of the protocol are executed exactly as specified in CPAL, CPAL-ES will determine if the goals given in the ASSERT statements are met in the protocol run. This does not provide a guarantee that a clever intruder will not be able to construct a run of the protocol that will compromise the protocol security. CPAL-ES allows us to make the assumptions and the protocol goals explicit, and gives them both a formal meaning. It also allows us to generate many different versions and runs of a protocol quickly, and provide a message based evaluation of what each run accomplishes.

6.3.2 Variations on the Needham and Schroeder Protocol

The Needham and Schroeder protocol was a watershed in cryptographic protocol development and is the basis for many other protocols that were designed either for slightly different purposes or to correct problems with it. In this section, we evaluate some of the variations on the Needham and Schroeder protocol using CPAL-ES.

6.3.2.1 Denning and Sacco

In [DS81] the authors go beyond pointing out the problem with the Needham and Schroeder protocol; they also propose their own private key protocol. As with the Needham and Schroeder protocol, their protocol begins with the first message going to the authentication server. The difference is that Denning and Sacco rely on the authentication

server and a timestamp to synchronize the run. In its only message, the authentication server presents a timestamp that each principal can use to ensure the freshness of the run. The CPAL-ES evaluation of the Denning and Sacco protocol is provided in Figure 6.6 and the final verification condition mirrors the Needham and Schroeder result. That is, the protocol meets its specified goals if the function used by principals A and B is the same.

6.3.2.2 Otway and Rees

The Otway and Rees protocol [OTWR87] given in Chapter 1, Figure 1.2, offered another alternative to the Needham and Schroeder Private Key Protocol. In this protocol the authors avoid the playback problem by including a nonce, C, which is used as a session identifier and is included with each message.

Additionally, Otway and Rees designed the protocol so the run originator sends the first message to the intended recipient rather than to the authentication server. This allows each participant to generate the nonce they will use to verify the freshness of each message in the run.

Denning and Sacco Private Key Protocol

```
S: =>A(S.kas);
A: <-(A.kas);
S: =>B(S.kbs);
B: <-(B.kbs);
A: => S(<A.A,A.B>);
S: <-(S.msg);
S: (S.A,S.B) := S.msg;
S: S.kab := new;
S: S.T1 := new;
S: => A(e[<S.B,S.kab,S.T1,e[<S.T1,S.kab,S.A>]S.kbs>]S.kas);
A: <-(A.Msg1);
A: A.tmp := d[A.Msg1]A.kas;
A: (A.dst,A.kab,A.T1,A.Ticket) := A.tmp;
A: assert((A.B == A.dst));
A: => B(A.Ticket);
B: <-(B.Ticket);
B: B.tmp1 := d[B.Ticket]B.kbs;
B: (B.T1,B.kab,B.A) := B.tmp1;
B: gassert((S.T1 == B.T1));
B: => A(e[B.N1]B.kab);
A: <-(A.Msg2);
A: A.N1 := d[A.Msg2]A.kab;
A: => B(e[A.f(A.N1)]A.kab);
B: <-(B.Msg3);
B: B.N1' := d[B.Msg3]B.kab;
B: assert((B.f(B.N1) == B.N1'));

 *** End of Protocol ***
```

```
(((TRUE  and (B.f(B.N1) ==
d[e[A.f(d[e[B.N1]d[d[e[<<A.A,A.B>.2,unique.v1,unique.v0,e[<unique.v0,unique.v1,<A.
A,A.B>.1>]S.kbs>]S.kas]S.kbsS.kas.4].2]d[e[<<A.A,A.B>.2,unique.v1,unique.v0,e[<uni
que.v0,unique.v1,<A.A,A.B>.1>]S.kbs>]S.kas]S.kas.2)]d[e[<<A.A,A.B>.2,unique.v1,un
ique.v0,e[<unique.v0,unique.v1,<A.A,A.B>.1>]S.kbs>]S.kas]S.kas.2]d[d[e[<<A.A,A.B>
.2,unique.v1,unique.v0,e[<unique.v0,unique.v1,S.kas<A.A,A.B>.1>]S.kbs>]S.kas].4]S.k
bs.2)) and (unique.v0 ==
d[d[e[<<A.A,A.B>.2,unique.v1,unique.v0,e[<unique.v0,unique.v1,<A.A,A.B>.1>]S.kbs
>]S.kas]S.kas.4]S.kbs.1)) and (A.B==
d[e[<<A.A,A.B>.2,unique.v1,unique.v0,e[<unique.v0,unique.v1,<A.A,A.B>.1>]S.kbs>]
S.kas]S.kas.1))
****** Simplified predicate follows.

(B.f(B.N1) == A.f(B.N1))
```

Figure 6.6

We give the CPAL-ES evaluation of the Otway and Rees protocol in Figure 6.7. Notice that since no function is used to protect the nonces, the only assumption that is required is that the previously held keys between the two participants and the authentication server are the same, which is again reflected in the first four statements of the protocol. Granted this assumption, the verification condition for the Otway and Rees protocol run as given reduces to TRUE and the protocol meets the stated goals.

Also notice that even though the Otway and Rees protocol requires only four transmissions in each protocol run, the size of the original verification condition in Figure 6.8, is substantially greater for the Otway and Rees protocol than it was for the previous examples. This results from the characteristics of the Otway and Rees protocol. In this protocol, there are several components to each message and operations on values are nested in most messages. The operations themselves add to the length of the protocol and, thus, to the size of the verification condition.

Otway and Rees Private Key Protocol

```
S: => A(S.kas);
A: <-(A.kas);
S: => B(S.kbs);
B: <-(B.kbs);
A: => B(<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]A.kas>);
B: <-(B.msg1);
B: (B.C,B.A',B.B',B.Ticket1) := B.msg1;
B: B.Ticket2 := e[<B.Nb,B.C,B.A',B.B'>]B.kbs;
B: => S(<B.C,B.A',B.B',B.Ticket1,B.Ticket2>);
S: <-(S.msg2);
S: (S.C,S.A,S.B,S.Ticket1,S.Ticket2) := S.msg2;
S: (S.Na,S.Ca,S.A',S.B') := d[S.Ticket1]S.kas;
S: assert((S.A' == S.A));
S: assert((S.B' == S.B));
S: assert((S.Ca == S.C));
S: (S.Nb,S.Cb,S.A'',S.B'') := d[S.Ticket2]S.kbs;
S: assert((S.Cb == S.C));
S: S.passa := e[<S.Na,S.kab>]S.kas;
S: S.passb := e[<S.Nb,S.kab>]S.kbs;
S: => B(<S.C,S.passa,S.passb>);
B: <-(B.msg3);
B: (B.C',B.passa,B.passb) := B.msg3;
B: assert((B.C == B.C'));
B: (B.Nb',B.kab) := d[B.passb]B.kbs;
B: assert((B.Nb == B.Nb'));
B: => A(<B.C,B.passa>);
A: <-(A.msg4);
A: (A.C',A.passa) := A.msg4;
A: assert((A.C == A.C'));
A: (A.Na',A.kab) := d[A.passa]A.kas;
A: assert((A.Na == A.Na'));

 *** End of Protocol ***
*** Original verification condition is in Figure 6.8
****** Simplified predicate follows.

TRUE
```

Figure 6.7

CPAL-ES Verification Condition for the Otway and Rees Protocol

(((((((TRUE and (A.Na == d[<<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.1,<<<A.C,A.A,A.B,
e[<A.Na,A.C,A.A,A.B>]S.kas>.1,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.2,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,
A.B>]S.kas>.3,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.4,e[<B.Nb,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.k
as>.1,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.2,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.3>]S.kbs>.1,e[
<d[<<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.1,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.2,<A.C,A.A,A.
B,e[<A.Na,A.C,A.A,A.B>]S.kas>.3,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.4,e[<B.Nb,<A.C,A.A,A.B,e[<A.N
a,A.C,A.A,A.B>]S.kas>.1,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.2,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.
kas>.3>]S.kbs>.4]S.kas.1,S.kab>]S.kas,e[<d[<<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.1,<A.C,A.A,A.B,e[<A.
Na,A.C,A.A,A.B>]S.kas>.2,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.3,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]
S.kas>.4,e[<B.Nb,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.1,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.2,
<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.3>]S.kbs>.5]S.kbs.1,S.kab>]S.kbs>.2>.2]S.kas.1)) and (A.C ==
<<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.1,<<<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.1,<A.C,A.A,A.B
,e[<A.Na,A.C,A.A,A.B>]S.kas>.2,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.3,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,
A.B>]S.kas>.4,e[<B.Nb,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.1,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.k
as>.2,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.3>]S.kbs>.1,e[<d[<<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.ka
s>.1,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.2,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.3,<A.C,A.A,A.
B,e[<A.Na,A.C,A.A,A.B>]S.kas>.4,e[<B.Nb,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.1,<A.C,A.A,A.B,e[<A.N
a,A.C,A.A,A.B>]S.kas>.2,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.3>]S.kbs>.4]S.kas.1,S.kab>]S.kas,e[<d[<<A
.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.1,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.2,<A.C,A.A,A.B,e[<A.
Na,A.C,A.A,A.B>]S.kas>.3,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.4,e[<B.Nb,<A.C,A.A,A.B,e[<A.Na,A.C,A
.A,A.B>]S.kas>.1,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.2,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.3>
]S.kbs>.5]S.kbs.1,S.kab>]S.kbs>.2>.1)) and (B.Nb == d[<<<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.1,
<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.2,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.3,<A.C,A.A,A.B,e[<
A.Na,A.C,A.A,A.B>]S.kas>.4,e[<B.Nb,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.1,<A.C,A.A,A.B,e[<A.Na,A.C
,A.A,A.B>]S.kas>.2,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.3>]S.kbs>.1,e[<d[<<A.C,A.A,A.B,e[<A.Na,A.C,
A.A,A.B>]S.kas>.1,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.2,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.
3,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.4,e[<B.Nb,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.1,<A.C,A
.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.2,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.3>]S.kbs>.4]S.kas.1,S.kab>]
S.kas,e[<d[<<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.1,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.2,<A.C,
A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.3,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.4,e[<B.Nb,<A.C,A.A,A.B,
e[<A.Na,A.C,A.A,A.B>]S.kas>.1,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.2,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,
A.B>]S.kas>.3>]S.kbs>.5]S.kbs.1,S.kab>]S.kbs>.3]S.kbs.1) and (<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.1 ==
<<<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>. 1,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.2,<A.C,
A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.3,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.4,e[<B.Nb,<A.C,A.A,A.B,
e[<A.Na,A.C,A.A,A.B>]S.kas>.1,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.2,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,
A.B>]S.kas>.3>]S.kbs>.1,e[<d[<<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.1,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A
.B>]S.kas>.2,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.3,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.4,e[<B.
Nb,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.1,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.2,<A.C,A.A,A.B,
e[<A.Na,A.C,A.A,A.B>]S.kas>.3>]S.kbs>.4]S.kas.1,S.kab>]S.kas,e[<d[<<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.ka
s>.1,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.2,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.3,<A.C,A.A,A.
B,e[<A.Na,A.C,A.A,A.B>]S.kas>.4,e[<B.Nb,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.1,<A.C,A.A,A.B,e[<A.N
a,A.C,A.A,A.B>]S.kas>.2,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.3>]S.kbs>.5]S.kbs.1,S.kab>]S.kbs>.1)) and
(d[<<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.1, <A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.2,<A.C,
A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.3,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.4,e[<B.Nb,<A.C,A.A,A.B,
e[<A.Na,A.C,A.A,A.B>]S.kas>.1,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.2,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,
A.B>]S.kas>.3>]S.kbs>.5]S.kbs.2 == <<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.1,<A.C,A.A,A.B,
e[<A.Na,A.C,A.A,A.B>]S.kas>.2,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.3,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,
A.B>]S.kas>.4,e[<B.Nb,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.1,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.k
as>.2,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.3>]S.kbs>.1)) and (d[<<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]
S.kas>.1,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S. kas>.2,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.3,
<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.4,e[<B.Nb,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.1,<A.C,A.A
,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.2,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.3>]S.kbs>.4]S.kas.2 ==
<<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.1,<A.C,A.A,A.B, e[<A.Na,A.C,A.A,A.B>]S.kas>.2,<A.C,A.A,A.B,
e[<A.Na,A.C,A.A,A.B>]S.kas>.3,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.4,e[<B.Nb,<A.C,A.A,A.B,e[<A.Na,
A.C,A.A,A.B>]S.kas>.1,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.2,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.k
as>.3>]S.kbs>.1)) and (d[<<A.C,A.A,A.B, e[<A.Na,A.C,A.A,A.B>]S.kas>.1,<A.C,A.A,A.B,e[<A.Na,A.C,
A.A,A.B>]S.kas>.2,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.3,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.
4,e[<B.Nb,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.1,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.2,<A.C,A
.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.3>]S.kbs>.4]S.kas.4 == <<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.1,
<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.2, <A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.3,<A.C,A.A,A.B,
e[<A.Na,A.C,A.A,A.B>]S.kas>.4,e[<B.Nb,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.1,<A.C,A.A,A.B,e[<A.Na,
A.C,A.A,A.B>]S.kas>.2,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.3>]S.kbs>.3)) and(d[<<A.C,A.A,A.B,
e[<A.Na,A.C,A.A,A.B>]S.kas>.1,<A.C,A.A,A.B,
e[<A.Na,A.C,A.A,A.B>]S.kas>.2,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,
A.B>]S.kas>.3,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.4,e[<B.Nb,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.k
as>.1,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.2,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.3>]S.kbs>.4]S.
kas.3 ==<<A.C,A.A,A.B,e[<A.Na,A.C, A.A,A.B>]S.kas>.1,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.2,
<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.3,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.4,e[<B.Nb,<A.C,A.A
,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.1,<A.C,A.A,A.B,e[<A.Na,A.C,A.A,A.B>]S.kas>.2,<A.C,A.A,A.B,e[<A.Na,A.C,
A.A,A.B>]S.kas>.3>]S.kbs>.2))

Figure 6.8

This protocol illustrates the requirement for an automated tool to evaluate verification conditions. As you see, the original verification condition for the Otway and Rees is more than a page long, and working through this type predicate manually would be almost futile. Moreover, we have seen other protocols that generate verification conditions that are much larger than this one. We believe that, like computer program verification, this level of effort is inherent in protocol verification. Automation of this effort is provided by CPAL-ES. We have not encountered any protocols whose verification condition does not reduce to a workable size once the simplifications in CPAL-ES are applied.

Since we illustrated the size and complexity of the original verification condition with the Otway and Rees protocol and because of the complex nature of these verification conditions, we will omit large, original verification conditions from subsequent examples.

6.3.2.3 Abadi and Needham's Modification to Otway and Rees

In [AN94], Abadi and Needham suggest modifications to the Otway and Rees protocol that result in a protocol that requires less encryption than the original version. They use the convention of attaching names to messages to produce this efficiency. We give the CPAL-ES evaluation of the Abadi and Needham version of the protocol in Figure 6.9.

The global assertion operator GASSERT is used in Figure 6.9 to show the desired relationship between each principal’s understanding of who the principals are. If B’s representation of the source (B.A) is the same as that of the source, the names introduced by Abadi and Needham accomplish their desired result.

Abadi and Needham Modification to the Otway and Rees Protocol

```
S: => A(S.kas);  A: <-(A.kas);
S: => B(S.kbs);  B: <-(B.kbs);
A: A.Na:=new;
A: => B(<A.A,A.B,A.Na>);
B: <-(B.msg1);
B: (B.A,B.B,B.Na) := B.msg1;
B: B.Nb:=new;
B: => S(<B.A,B.B,B.Na,B.Nb>);
S: <-(S.msg2);
S: (S.A,S.B,S.Na,S.Nb) := S.msg2;
S: S.kab:=new;
S: S.passa := e[<S.Na,S.A,S.B,S.kab>]S.kas;
S: S.passb := e[<S.Nb,S.A,S.B,S.kab>]S.kbs;
S: => B(<S.passa,S.passb>);
B: <-(B.msg3);
B: (B.passa,B.passb) := B.msg3;
B: (B.Nb',B.A,B.B,B.kab) := d[B.passb]B.kbs;
B: assert(B.Nb == B.Nb');
B: gassert(B.A == A.A);
B: gassert(B.B == A.B);
B: => A(B.passa);
A: <-(A.passa);
A: (A.Na',A.A,A.B,A.kab) := d[A.passa]A.kas;
A: assert(A.Na == A.Na');
A: gassert(B.A == A.A);
A: gassert(B.B == A.B);
Global: gassert(B.kab == A.kab);
*** *End of Protocol.
**** Simplified predicate follows.
TRUE
```

Figure 6.9

6.3.2.4 Kerberos

The Kerberos Authentication System [MNSS87] is a broadly implemented application using a variation of the Needham and Schroeder Private Key Protocol as its primary authentication protocol. The CPAL-ES evaluation of the Kerberos protocol is given in figure 6.10.

The Kerberos protocol is unlike previously evaluated protocols in that Kerberos utilizes timestamps rather than nonces to guarantee message freshness. The Kerberos

developers compare the value of the timestamp received against the value of a local, synchronized clock within some given variance. In the given CPAL-ES Kerberos evaluation, we utilize the global ASSERT statement to reflect the expectation of agreement between the times of the different principals.

Because Kerberos utilizes a function of the timestamp, the final result of the CPAL-ES evaluation is to show that the protocol works if we assume that the two principals utilize the same function g.

The Kerberos Private Key Protocol

```
S: => A(S.kas);
A: <-(A.kas);
S: => B(S.kbs);
B: <-(B.kbs);
A: => S(<A.A,A.B>);
S: <-(S.msg);
S: (S.A,S.B) := S.msg;
S:=>A(e[<S.Ts,S.L,S.kab,S.B,e[<S.Ts,S.L,S.kab,S.A>]S.kbs>]S.kas);
A: <-(A.msg);
A: (A.Ts,A.L,A.kab,A.B,A.msg2) := d[A.msg]A.kas;
A: gassert(S.Ts == A.Ts);
A: => B(<A.msg2,e[<A.A,A.Ta>]A.kab>);
B: <-(B.msg);
B: (B.msg1,B.msg2) := B.msg;
B: (B.Ts,B.L,B.kab,B.A) := d[B.msg1]B.kbs;
B: gassert(S.Ts == B.Ts);
B: (B.A,B.Ta) := d[B.msg2]B.kab;
B: => A(e[B.g(B.Ta)]B.kab);
A: <-(A.msg);
A: A.Ta' := d[A.msg]A.kab;
A: assert(A.g(A.Ta) == A.Ta');
 *** End of Protocol ***
****** Simplified predicate follows.

(A.g(A.Ta) == B.g(A.Ta))
```

Figure 6.10

6.4 Active Attacks on Cryptographic Protocols.

The characteristic of BAN Logic that Nessett identified in his criticism in [NESS90] was that BAN Logic provides only a static evaluation of any protocol. As we briefly addressed in Chapter 2, by static evaluation we mean that the evaluation is valid only if the steps of the protocol are accomplished in exactly the sequence and with exactly the content of messages as specified. Clearly, accomplishment of static evaluation of protocols by a formal method is very important, since any protocol that does not meet its goals under static evaluation is useless in practice. As Burrows, et. al. point out in their rejoinder to Nessett [BAN90b], static evaluation is just one step in evaluating a cryptographic protocol. BAN Logic accomplishments aside, the weakness remains, i.e., the evaluation makes no provision to consider active attacks by persistent intruders.

While a passive attack is accomplished by an intruder listening to the messages in a communication without interfering, an active attack is characterized by the intruder intervening in a communication by intercepting messages that may never be delivered, copying messages for later replay either with or without modification, and generating new messages for insertion into a communication session. The goal of an intruder is to spoof a valid participant. A spoof causes a principal A to have an erroneous belief, such as A believing that A has a established a secure communication session with another principal B, when in fact A has not established a session with B. One type of spoof may cause A to have such a belief when in fact no session is established, while another spoof may have A establish a session with a principal other than B, maybe the intruder. Other spoofs involve beliefs regarding the characteristics of keys or maybe the status of an expected secret.

6.4.1 Denning and Sacco Attack on Needham and Schroeder Private Key Protocol

The most difficult characteristic regarding spoofs is that they come in an infinite assortment of types. Regardless of how many different spoofs a protocol protects against,

it is impossible to guarantee that an intelligent intruder will not discover a new one that defeats the protocol. Current verification methods center on detecting known types of spoofs, such as the classic replay attack given by Denning and Sacco in [DS81].

The idea of a replay attack is to record a valid protocol session in order to use one or more messages or message components in a later session. The need to use a recorded message stems from the nature of cryptography, i.e. that the intruder cannot generate the value expected in a protocol without first possessing the necessary key. In order to circumvent this problem, the intruder does not attempt to generate the necessary encrypted value, but rather steals the value from someone that has the key and passed a suitably structured value in a previous session.

To illustrate this point, consider the attack Denning and Sacco constructed on the Needham and Schroeder Private Key Protocol [NS78] given in Figure 1.1 and again along with the attack session in Figure 6.11. The intent of this protocol is to allow two principals to utilize a trusted key server that they share separate secret keys with, to establish a new common session key suitable for use in a secure communication session. At the end of the protocol, the goal is for both A and B to have the key (represented by “kab”) in their private address space and for no one else except the trusted server to know that key (i. e. have the key in their private address space).

Statically, the Needham and Schroeder Private Key Protocol seems to meet its goals. If the protocol steps are executed as written, there is no clear way that a passive intruder could gain access to the session key for an ongoing session. The same cannot be said about an active intruder.

In the attack sequence given in Figure 6.11, notice that in the first session, the intruder copies the third message in the protocol (steps 8 and 9), then lets the message pass unchanged to its intended recipient. The assumption is that the attacker may compromise

key kab at some point, possibly long after the session it was used in is over. This assumption is credible in that it is recognized that keys become more vulnerable over time, particularly if a sizable amount of ciphertext is available for analysis by the intruder. Since kab is a session key, it is fair to assume that a large amount of ciphertext will be available. Once the key is compromised and the intruder possesses kab, the intruder patiently waits

Replay Attack on the Needham and Schroeder Private Key Protocol

	Session 1	Session 2
1.	A: =>S(<A,B,na>);	A: =>S:(<A,B,na'>);
2.	S: <-(msg1);	S: <-(msg1‘);
3.	S: (A,B,na) := msg1;	S: (A,B,na’) := msg1‘;
4.	S:=>A(e[<na,B,kab,e[kab,A]kbs>]kas);	S:=>A(e[<na’,B,kab’,e[kab’,A]kbs>]kas);
5.	A: <-(msg2);	A: <-(msg2‘);
6.	A: (na,B,kab,msg3) := msg2;	A: (na’,B,kab’,msg3‘) := msg2‘;
7.	A: ->B(msg3);	A: ->B(msg3‘);
8.	I: <-(msg3);	I: <-(msg3‘);
9.	I: =>B(msg3);	I: =>B(msg3);
10.	B: <-(msg3);	B: <-(msg3);
11.	B: (kab,A) := msg3;	B: (kab,A) := msg3;
12.	B: nb := new;	B: nb’ := new;
13.	B: =>A(e[nb]kab);	B: =>A(e[nb’]kab);
14.	A: <-(msg4);	A: <-(msg4’);
15.	A: nb := d[msg4]kab;	A: nb := d[msg4’kab;
16.	A: =>B:(e[nb-1]kab);	A: =>B:(e[nb’-1]kab);
17.	B: <-(msg5);	B: <-(msg’5);
18.	B: nb" := d[msg5]kab;	B: nb" := d[msg5’]kab;
19.	B: assert((nb-1) == nb");	B: assert((nb’-1) == nb");

Figure 6.11

for principal A to attempt another session with B, which is easily detected.

The first seven steps of the second session occur as intended with kab' and na' signifying a new key and new nonce for the second session respectively. The third message of the attack session, the message from A to B in step 7, is intercepted by the intruder. The intruder then substitutes the message from the previous session in step 8 of the current ses-

sion. When B receives this message in step 10, there is no reason to doubt its originality. The second session progresses to completion with B believing the old session key kab, decrypted in step 11, is being established as the session key for the new session. The intruder is now in a possession to intercept all messages originating from principal A and conduct the session with principal B as though the intruder were A himself.

6.4.2 Parallel Session Attacks

Bird, et. al. illustrated another method active attackers may use in [BIRD93]. They showed how an intruder may use the parallel nature of systems in order to acquire an encrypted value they can use in a protocol attack. Using what is called a parallel session attack, given a suitable protocol the intruder will manipulate one or more principals so that at least one principal is running more than one cryptographic protocol session concurrently. The intruder then copies messages from one session into an appropriate place in the other session.

The next two examples illustrate how having one principal of a protocol session engaged in a second concurrent protocol session can allow an intruder to spoof one or both principals. We use the convention of indenting actions in the attack session for readability.

6.4.3 Bird et. al Attack on the ISO Protocol

We briefly mentioned earlier that Bird et. al, constructed an attack on the proposed ISO protocol in [BIRD93]. The attack they constructed is a parallel session attack. The complete sequence of that attack is given in Figure 6.12. In this attack, the intruder uses principal A to get an encrypted value to be used later to spoof A. Recall that in the ISO protocol, the first message contains only a nonce. The recipient encrypts the nonce, catenates it with a new nonce (also encrypted) and returns the final form of the message to the session originator.

In the attack sequence, the intruder intercepts A's originating message to B (action 4), then originates a parallel session with A using A's nonce to initiate the sequence in action 5. In A's response to the intruder's request for the parallel session in action 8, the original nonce is encrypted under the key shared by A and B, kab. The intruder then uses that encrypted nonce to complete the original session begun by A as shown in action 10. At the assert statement in action 15, A believes a valid session is begun with B, which is not true. In fact, no session is begun, because the intruder does not know the value of kab. Nonetheless, the protocol has been defeated and A was spoofed into having a false belief.

The attack created when an intruder passes a value to a principal for the purpose of attaining the encryption of that value under a specific secret key so that the encrypted value may be used in a spoof is known as an ***oracle attack***. The duped principal effectively acts as an oracle for the intruder, allowing the intruder to attain encrypted values without knowing the encryption key. The ISO attack in Figure 6.12 is an oracle attack constructed using a parallel session.

Notice that in the attack sequence of Figure 6.12, the stated goals of principal A in both protocol sessions are met in a way not intended by the protocol creator. Principal A is spoofed by the intruder that cleverly uses data copied from the parallel session to meet the requirements of the protocol. Thus, even though the message-based goals are met, any goal based on the equivalence of these nonces, such as "A believes that B must have sent Nb" are not met.

In the ISO parallel session attack example, CPAL-ES quickly and reliably determined that this sequence of interactions between principals met the message based goals specified for a given [compromised] run of a protocol. The global assert statement in step 21 reflects one characteristic of this compromise, i.e. that I "knows" the value Na2 that is needed to complete the protocol and that A believes only B can know Na2. The "TRUE" result of the evaluation reflects the fact that the specified run of the protocol meets its

desired objectives. Since we know the steps violate the desired results, the TRUE result indicates we have found an attack on the protocol.

ISO Protocol Attack from [BIRD93]

```
1a      A: =>B(k);
1b      B: <-(k);
2       A: Na := new;
3       A: ->B(Na);
4       I: <-(Na);
5           I: =>A(Na); -- Intruder initiates a parallel session with A
6           A: <-(Na);
7           A: Na2 := new;
8           A: ->B(<e[Na]k,e[Na2]k>);
9           I: <-(msg1);
10          I: =>A(msg1); --I uses data from parallel session in 1st session
11      A: <-(msg);
12      A: (Nae,Nbe) := msg;
13      A: Na' := d[Nae]k;
14      A: Nb := d[Nbe]k;
15      A: assert(Na' == Na);
16      A: ->B(Nb);
17      I: <-(Na2');
18          I: =>A(Na2');
19          A: <-(Na2');
20          A: assert(Na2' == Na2);
21      x: gassert(I.Na2' == A.Na2);

(((TRUE  and (unique.v0 == d[<e[unique.v1]A.k,e[unique.v0]A.k>.2]A.k)) and
    (unique.v0 == d[<e[unique.v1]A.k,e[unique.v0]A.k>.2]A.k)) and (unique.v1 ==
    d[<e[unique.v1]A.k,e[unique.v0]A.k>.1]A.k))
****** Simplified predicate follows.

TRUE
```

Figure 6.12

6.4.4 The Trivial Two-way Protocol Attack

In [BIRD92], the authors give a trivial two-way authentication protocol and an attack against that protocol. The protocol itself is given in Appendix D. The attack the

authors construct on this protocol is a parallel session attack. In Figure 6.13, Bird's attack on the trivial two-way protocol illustrates how the stated message-based goal of the original protocol may be met, but without guaranteeing the intent of the original protocol.

Attack on the BIRD Trivial Two-way Protocol

```
S: => A(kas);
A: <-(kas);
S: => B(kbs);
B: <-(kbs);
A: ->B(e[<A,e[N1]k>]k);
I: <-(msg1);
-- Above here is from a previous session.
-- The intruder now initiates a second session with B.
        I: =>B(msg1);
        B: <-(msg1);
        B: (src,msg2) := d[msg1]k;
        B: N1 := d[msg2]k;
        B: msg3 := e[<B,e[N2]k>]k;
        B: ->A(<N1,msg3>);
                I: <-(msg3a);-- The intruder initiates a session with A using the
                    encrypted msg sent from B.
                I: (N1,msg3) := msg3a;
                I: =>A(msg3);
                A: <-(msg3);
                A: (dst,msg4) := d[msg3]k;
                A: N'1 := d[msg4]k;
                A: msg5 := e[<A,e[N3]k>]k;
                A: ->B(<N'1,msg5>);
                I: <-(msg6);
        I: (N2,msg5) := msg6; -- I now uses the information decrypted by A to spoof B.
        I: =>B(N2);
        B: <-(N'2);
        B: assert(N'2 == N2);
 *** End of Protocol ***

(TRUE and (B.N2 == <d[d[<d[d[e[<A.A,e[A.N1]A.k>]A.k]A.k.2]A.k,
e[<B.B,e[B.N2]A.k>]A.k>.2]A.k.2]A.k,e[<A.A,e[A.N3]A.k>]A.k>.1))
    ****** End of  verification condition.  Simplified predicate follows.******
    TRUE
```

Figure 6.13

6.4.5. Abadi and Needham Attack on the Woo and Lam Protocol.

In [AN94] Abadi and Needham give an attack on the Woo and Lam protocol given earlier in Figure 6.3. The focus of their attack is the connection between the messages in the protocol. By starting two concurrent sessions with principal B and impersonating principal A in one of the sessions, the intruder is able to substitute the final response in one session for the final response in the other session. The end result is to spoof B into believing B has a valid session with A, when A is not involved in the session.

Attack on the Woo and Lam Protocol

```
S: => A(kas);   A: <-(kas);
S: => B(kbs);    B: <-(kbs);
S: => I(kis);      B: <-(kis);

 I: =>B(A);          -- I initiates a session with B posing as A
B: <-(A);
B: ->A(Nb1);        -- B's first response to first session
I: <-(Nb1);         -- I intercepts B's msg
                I: =>B(I);        -- I initiates a second session with B on its own behalf
                B: <-(I);
                B: ->I(Nb2);    -- B's response to second session with a new nonce
                I: <-(Nb2);
                I: =>B(e[Nb1]kis); -- I subsitutes B's nonce from the first session
B: <-(msg1);-- B cannot detect whether this message is from the first or second session
I: =>B(e[Nb1]kis);
B: =>S(e[<A,msg1>]kbs);
S: <-(msg1a);
S: (A,msg1) := d[msg1a]kbs;
S: Nb1 := d[msg1]kas;
                B: <-(msg2);    -- msg1 from the first session is equal to msg2
                B: =>S(e[<I,msg2>]kbs);
                S: <-(msg2a);
                S: (I,msg2) := d[msg2a]kbs;
                S: Nb1' :=d[msg2]kis;           --If src==I in last msg use kis
                S: =>B(e[Nb1]kbs);
B: <-(msg5);-- B cannot detect whether this message is from the first or second session
B: Nb1' := d[msg5]kbs;
S: =>B(e[Nb1']kbs);
                B: <-(msg6);
                B: Nb1" := d[msg6]kbs;
B: assert(Nb1' == Nb1);
-- Because msgs are async without guaranteed origin, B believes this is from A.
 *** End of Protocol ***

(TRUE and (B.Nb1==d[e[d[d[e[<I.I,e[B.Nb1]S.kis>]S.kbs]S.kbs.2]S.kis]S.kbs]S.kbs))

****** Simplified predicate follows

TRUE
```

Figure 6.14

The CPAL-ES evaluation given in Figure 6.14 captures the nature of the attack on the Woo and Lam protocol. Since the intruder actions will not complete the second ses-

sion, no assertions are made about the second session's validity. The attack is successful because the intruder is able to substitute a response intended for the attack session into the original session.

6.4.6 CPAL-ES and Intruder Actions

Evaluation of the replay attack in Figure 6.11 and the parallel session attacks of Figure 6.12, 6.13, and 6.14 clearly illustrates how CPAL-ES gives explicit meaning to the actions an intruder may take. Represented as routine network operations, the intruder actions shown in these attacks are straightforward and well-defined. An intruder may receive a message into its personal address space, modify data in the address space, and submit messages onto the net. With these three capabilities, which closely mirror the capabilities intruders are routinely assumed to have, the intruder can accomplish a myriad of attacks. The meaning of these attacks is formally defined by CPAL-ES.

The CPAL-ES model is accurate in that it does not allow the intruder to directly change a valid principal's address space. Rather, such changes are forced to occur indirectly, across the network, using the normal send and receive statements. Because the CPAL-ES view of the intruder is simple and intuitive, an analyst can easily utilize CPAL-ES to speculate about possible intruder actions and quickly evaluate the results of those actions. Because intruder actions are explicit in this way, they are easier to estimate and explore.

6.5 Evaluating Public-Key Protocols

The protocols and attacks we considered so far are based on symmetric key, or private key, protocols. CPAL-ES also allows us to evaluate asymmetric, or public, key protocols. This is accomplished through the encrypt public (ep) and decrypt public (dp) operators.

The first public key protocol we look at is the Needham and Schroeder Public key protocol [NS78]. The CPAL-ES evaluation is given in Figure 6.15. The intent of the Needham and Schroeder Public Key Protocol is to allow a trusted authentication server to securely distribute the public key of two parties to each other so a secure session may be accomplished using each participant's public key.

Needham and Schroeder Public Key Protocol

```
S: assume(global.decrypt(B.ks+,S.ks-));
S: assume(global.decrypt(A.ks+,S.ks-));
A: => S(<A.A,A.B>);
S: <-(S.msg);
S: (S.A,S.B) := S.msg;
S: => A(ep[<S.kb+,S.B>]S.ks-);
A: <-(A.msg);
A: (A.kb+,A.B) := dp[A.msg]A.ks+;
A: => B(ep[<A.Na,A.A>]A.kb+);
B: <-(B.msg);
B: (B.Na,B.A) := dp[B.msg]B.kb-;
B: => S(<B.B,B.A>);
S: <-(S.msg2);
S: (S.B,S.A) := S.msg2;
S: => B(ep[<S.ka+,S.A>]S.ks-);
B: <-(B.msg2);
B: (B.ka+,B.A) := dp[B.msg2]B.ks+;
B: => A(ep[<B.Na,B.Nb>]B.ka+);
A: <-(A.msg2);
A: (A.Na',A.Nb) := dp[A.msg2]A.ka-;
A: assert((A.Na == A.Na'));
A: => B(ep[A.Nb]A.kb+);
B: <-(B.msg3);
B: B.Nb' := dp[B.msg3]B.kb-;
B: assert((B.Nb == B.Nb'));

 *** End of Protocol ***
****** Simplified predicate follows.

((not (global.decrypt(B.kb-,S.kb+)) or ((not (global.decrypt(A.ka-,S.ka+)) or
global.decrypt(B.kb-,S.kb+)) and global.decrypt(A.ka-,S.ka+))) and global.decrypt(B.kb-
,S.kb+))
```

Figure 6.15

CPAL allows the normal naming convention of suffixing a plus sign to the key name for the public key and suffixing a minus sign for the private key of the pair. Rather than depend on this convention to identify the public and private key pair, CPAL-ES matches the public and private key pairs in the evaluation and shows us what the relationships must be in order for the protocol to execute properly. To illustrate the use of the ASSUME statement with public key protocols, the run of the Needham and Schroeder Public Key Protocol shown in Figure 6.15 was executed without two ASSUME statements. The resulting predicate illuminates the additional relationships that must hold in order for the protocol to work as desired. By adding the ASSUME statements:

S: assume(global.decrypt(B.kb-,S.kb+));

S: assume(global.decrypt(A.ka-,S.ka+));

the verification condition simplifies to TRUE.

The Tatebashi, Matsuzaki, and Newman (TMN) protocol [TMN91] is another cryptographic protocol that utilizes asymmetric key cryptography. It is particularly interesting because it uses a combination of symmetric and asymmetric key methods in order to pass a key to be used in a symmetric key secure communication session.

In the protocol, A and B secure their messages to S using S's public key which they are assumed to know. S secures its message to A using the temporary symmetric key supplied by A. B generates the key to be used as the session key. The CPAL-ES evaluation of the TMN protocol is given in Figure 6.16.

TMN Public Key Protocol

```
S: assume(global.decrypt(S.pks-,A.pks));
S: assume(global.decrypt(S.pks-,B.pks));
A: => S(<A.A,A.B,ep[A.ka]A.pks>);
S: <-(S.msg1);
S: (S.A,S.B,S.msg2) := S.msg1;
S: S.ka := dp[S.msg2]S.pks-;
S: => B(S.A);
B: <-(B.A);
B: => S(ep[B.kab]B.pks);
S: <-(S.msg3);
S: S.kab := dp[S.msg3]S.pks-;
S: => A(e[S.kab]S.ka);
A: <-(A.msg1);
A: A.kab := d[A.msg1]A.ka;
A: => B(e[A.important_data]A.kab);
B: <-(B.msg1);
B: B.important_data := d[B.msg1]B.kab;
X: gassert((A.important_data == B.important_data));

 *** End of Protocol ***

(TRUE  and (A.important_data == d[e[A.important_data]d[e[dp[ep[B.kab]B.pks]S.pks-
]dp[<A.A,A.B,ep[A.ka]A.pks>.3]S.pks-]A.ka]B.kab))(((d[e[dp[ep[B.kab]B.pks]S.pks-
]dp[ep[A.ka]A.pks]S.pks-]A.ka == B.kab) imp (A.important_data == A.important_data))
and (not ((d[e[dp[ep[B.kab]B.pks]S.pks-]dp[ep[A.ka]A.pks]S.pks-]A.ka == B.kab))
 imp (A.important_data == unique.v0)))(((dp[ep[A.ka]A.pks]S.pks- == A.ka) imp (B.kab
== B.kab)) and (not ((dp[ep[A.ka]A.pks]S.pks- == A.ka))  imp (unique.v2 == B.kab)))

      ****** Simplified predicate follows.

      TRUE
```

Figure 6.16

In [SIMM85], Simmons reports that there are flaws in the TMN protocol. In [KMM93] Kemmerer, Meadows, and Millen illustrate their respective cryptographic protocol analysis tools by demonstrating how their respective methods would detect one of the flaws that Simmons already discovered. We give one attack on the TMN protocol in Figure 6.17. Notice that the global assert statement reflects the knowledge that the intruder has of the “important data’.

Attack on the TMN Protocol

```
S: assume(global.decrypt(S.pks-,A.pks));
S: assume(global.decrypt(S.pks-,I.pks));
A: => S(<A.A,A.B,ep[A.ka]A.pks>);
S: <-(S.msg1);
S: (S.A,S.B,S.msg2) := S.msg1;
S: S.ka := dp[S.msg2]S.pks-;
S: -> B(S.A);
I: <-(I.A);
I: I.kab := new;
I: => S(ep[I.kab]I.pks);
S: <-(S.msg3);
S: S.kab := dp[S.msg3]S.pks-;
S: => A(e[S.kab]S.ka);
A: <-(A.msg1);
A: A.kab := d[A.msg1]A.ka;
A: -> B(e[A.important_data]A.kab);
I: <-(I.msg1);
I: I.important_data := d[I.msg1]I.kab;
X: gassert((I.important_data == A.important_data));

*** End of Protocol ***

(TRUE and
(d[e[A.important_data]d[e[dp[ep[unique.v0]I.pks]S.pks-
dp[<A.A,A.B,ep[A.ka]A.pks>.3]S.pks-]A.ka]unique.v0 ==
A.important_data))(((d[e[dp[ep[unique.v0]I.pks]S.pks-]dp[ep[A.ka]A.pks]S.pks-]A.ka
== unique.v0) imp (A.important_data == A.important_data)) and
(not ((d[e[dp[ep[unique.v0]I.pks]S.pks-]dp[ep[A.ka]A.pks]S.pks-]A.ka == unique.v0))
 imp (unique.v1 == A.important_data)))(((dp[ep[A.ka]A.pks]S.pks- == A.ka) imp
(unique.v0 == unique.v0)) and(not ((dp[ep[A.ka]A.pks]S.pks- == A.ka))
 imp (unique.v3 == unique.v0)))

****** Simplified predicate follows.

TRUE
```

Figure 6.17

The CCITT protocol [CCITT], [BAN89], [GS91] exercises CPAL-ES against another mix of cryptographic techniques used to establish a secure communication ses-

sion. Timestamps are used to ensure message freshness and nonces to connect messages in the session. Asymmetric key techniques are used to sign some message components and to authenticate others. The CPAL-ES evaluation of the CCITT X.509 protocol is given in Figure 6.18.

CCITT X.509 Authentication Protocol

```
-- Ta and Tb are timestamps
-- Xa, Ya, Xb, and Yb are user data.
-- The protocol ensures the integrity of Xa and Xb and
--    guarantees the privacy of Ya and Yb.
A: assume(global.decrypt(B.ka+,A.ka-));
B: assume(global.decrypt(A.kb+,B.kb-));
A: => B(<A.A,ep[<A.Ta,A.Na,A.B,A.Xa,ep[A.Ya]A.kb+>]A.ka->);
B: <-(B.recd1);
B: (B.A,B.msg1) := B.recd1;
B: (B.Ta,B.Na,B.dst,B.Xa,B.msg2) := dp[B.msg1]B.ka+;
B: assert(B.time_ok(B.Ta));
B: B.Ya := dp[B.msg2]B.kb-;
B: B.Nb := new;
B: => A(<B.B,ep[<B.Tb,B.Nb,B.A,B.Na,B.Xb,ep[B.Yb]B.ka+>]B.kb->);
A: <-(A.recd1);
A: (A.dst',A.msg3) := A.recd1;
A: (A.Tb,A.Nb,A.A,A.N'a,A.Xb,A.msg4) := dp[A.msg3]A.kb+;
A: assert(A.time_ok(A.Tb));
A: A.Yb := dp[A.msg4]A.ka-;
A: assert((A.Na == A.N'a));
A: => B(ep[A.Nb]A.ka-);
B: <-(B.msg5);
B: B.N'b := dp[B.msg5]B.ka+;
B: assert((B.Nb == B.N'b));

 *** End of Protocol.   Simplified predicate follows***

(A.time_ok(B.Tb) and B.time_ok(A.Ta))
```

Figure 6.18

In [BAN89] Burrows, Abadi and Needham construct an attack on the CCITT protocol. We give the CPAL-ES evaluation of that attack in Figure 6.19. The attack the authors construct is a sophisticated oracle attack using a both a replay session and parallel session. The intruder initiates a session with principal B using the message from a previous session. This allows B to decrypt the message under principal A's public key and find the destination identifier B in the message text. B responds with the appropriate message, which contains just enough cleartext information to allow the intruder to prompt principal A in the attack session to generate the exact value necessary to convince B that the intruder is principal A in the original session.

As we see, this attack illustrates many different, complex concepts surrounding attacks on cryptographic protocols. This one attack utilizes replay, parallel session, and oracle attack methods; it contains timestamps and nonces; and it is a public key protocol. Nonetheless, the CPAL-ES evaluation for this attack was easily constructed and effectively illuminates each of the concepts just discussed.

Attack on the CCITT X.509 Authentication Protocol

```
-- Ta and Tb are timestamps.  Xa, Ya, Xb, and Yb are user data.
-- The protocol ensures the integrity of Xa and Xb and guarantees the privacy of Ya
     and Yb.

A: assume(global.decrypt(B.ka+,A.ka-));
I: assume(global.decrypt(I.ki-,A.ki+));
A: assume(global.decrypt(I.ka+,A.ka-));
I: assume(global.decrypt(I.kb+,B.kb-));
A: ->B(<A,ep[<Ta,Na,B,Xa,ep[Ya]kb+>]ka->);
I: <-(msg1);
     -- Above here is from a previous session
I: =>B(msg1);
B: <-(recd1);
B: (A,msg1) := recd1;
B: (Ta,Na,dst,Xa,msg2) := dp[msg1]ka+;  --B does not check the timestamp Ta
B: Ya := dp[msg2]kb-;
B: ->A(<B,ep[<Tb,Nb,A,Na,Xb,ep[Yb]ka+>]kb->);
I: <-(recd1);
I: (B,msg3) := recd1;
I: (Tb,Nb,A,N'a,Xb,msg4) := dp[msg3]kb+;
A: =>I(<A,ep[<Ta2,Na2,I,Xa2,ep[Ya2]ki+>]ka->);
I: <-(recd2);
I: (A,msg5) := recd2;
I: (Ta2,Na2,B,Xa2,msg6) := dp[msg5]ka+;
I: =>A(<I,ep[<Ti,Nb,C,Na2,Xi,ep[Yi]ka+>]ki->);
A: <-(recd1);
A: (I,msg7) := recd1;
A: (Ti,Nb,A,N'a2,Xc,msg8) := dp[msg7]ki+;
A: Yb := dp[msg4]ka+;
A: assert(N'a2 == Na2);
A: =>I(ep[Nb]ka-);
I: <-(msg9);
I: =>B(msg9);
B: <-(msg5);
B: N'b := dp[msg5]ka+;
B: assert(N'b == Nb);
 *** End of Protocol.   Simplified predicate follows. ***

TRUE
```

Figure 6.19

6.6 Other protocols

In [SNEK92], Snekkenes gives a straitforward variation on the Needham and Schroeder Private Key Protocol [NEED78] that he calls KP. KP is not vulnerable to replay attack due to the nonce in each transmission. As the CPAL-ES evaluation in Figure 6.20 shows, the nonces transfer as desired. Nonetheless, the protocol is vulnerable to attack, as we see in the Figure 6.21.

Snekkenes' KP Protocol

```
S: => A(S.kas);
A: <-(A.kas);
S: => B(S.kbs);
B: <-(B.kbs);
A: => B(<A.A,A.Na,A.B>);
B: <-(B.msg);
B: (B.A,B.Na,B.B) := B.msg;
B: => S(<B.A,B.Na,B.B,B.Nb>);
S: <-(S.msg);
S: (S.A,S.Na,S.B,S.Nb) := S.msg;
S: => B(e[<S.kab,S.A,S.Nb,e[<S.A,S.B,S.Na,S.kab>]S.kas>]S.kbs);
B: <-(B.msg2);
B: (B.kab,B.A,B.Nb',B.msg3) := d[B.msg2]B.kbs;
B: assert(B.Nb' == B.Nb);
B: => A(<B.msg3,e[<B.Na,B.Nc,B.B>]B.kab>);
A: <-(A.msg);
A: (A.msg3,A.msg4) := A.msg;
A: (A.A,A.B,A.Na',A.kab) := d[A.msg3]A.kas;
A: assert(A.Na' == A.Na);
A: (A.Na,A.Nc,A.B) := d[A.msg4]A.kab;
A: => B(e[<A.Nc,A.A>]A.kab);
B: <-(B.msg4);
B: (B.Nc',B.A) := d[B.msg4]B.kab;
B: assert(B.Nc' == B.Nc);
 *** End of Protocol , simplified predicate follows.

TRUE
```

Figure 6.20

Due to the construction of the messages in KP and the nesting of encrypted components within messages, the original verification condition is complex, taking two pages to display. CPAL-ES quickly determines the truth of the verification condition with only two assumptions.

Snekkenes is able to compromise KP by attacking the message structure of the protocol. Because of the similar composition of the messages, Snekkenes is able to substitute the fourth message of the original session as the third message of an attack session without detection. Though the attack session does not reach completion, this spoof allows the intruder to gain access to the session key of the original session.

Attack on the KP Protocol

```
S: => A(S.kas);
A: <-(A.kas);
S: => B(S.kbs);
B: <-(B.kbs);
A: -> B(<A.A,A.Na,A.B>);
I: <-(I.msg);
I: (I.A,I.Na,I.B) := I.msg;
I: => A(<I.B,I.Ni,I.A>);
A: <-(A.msg);
A: (A.B,A.Ni,A.A) := A.msg;
A: -> S(<A.B,A.Ni,A.A,A.Na2>);
I: <-(I.msg1);
I: (I.B,I.Ni,I.A,I.Na2) := I.msg1;
I: => B(<I.A,I.Na2,I.B>);
B: <-(B.msg);
B: (B.A,B.Na2,B.B) := B.msg;
B: => S(<B.A,B.Na2,B.B,B.Nb>);
S: <-(S.msg);
S: (S.A,S.Na2,S.B,S.Nb) := S.msg;
S: => B(e[<S.kab,S.A,S.Nb,e[<S.A,S.B,S.Na2,S.kab>]S.kas>]S.kbs);
B: <-(B.msg2);
B: (B.kab,B.A,B.Nb',B.ticket_for_A) := d[B.msg2]B.kbs;
B: assert((B.Nb' == B.Nb));
B: -> A(<B.ticket_for_A,e[<B.Na2,B.Nc,B.B>]B.kab>);
I: <-(I.msg2);
I: (I.ticket_for_A,I.confirmation) := I.msg2;
I: => A(I.ticket_for_A);
A: <-(A.msg2);
A: (A.kab',A.B,A.Na2',A.ticket_for_B) := d[A.msg2]A.kas;
A: assert((A.Na2' == A.Na2));
A: -> B(A.ticket_for_B);
I: <-(I.kab);
I: (I.Na2,I.Nc,I.B) := d[I.confirmation]I.kab;
I: => B(e[<I.Nc,I.A>]I.kab);
B: <-(B.msg4);
B: (B.Nc',B.A) := d[B.msg4]B.kab;
B: assert((B.Nc' == B.Nc));
 *** End of Protocol, simplified predicate follows.
TRUE
```

Figure 6.21

The CPAL-ES evaluation of the protocol attributed to Yahalom in [CARL94] is given in Figure 6.22. The Yahalom Protocol is a three party, symmetric key protocol

whose goal is to provide two communicating principals with a symmetric key suitable for a secure session. The protocol requires four transmissions.

Yahalom Protocol

```
S: => A(S.kas);
A: <-(A.kas);
S: => B(S.kbs);
B: <-(B.kbs);
A: => B(<A.A,A.Na>);
B: <-(B.msg);
B: (B.A,B.Na) := B.msg;
B: => S(<B.B,e[<B.A,B.Na,B.Nb>]B.kbs>);
S: <-(S.msg);
S: (S.dst,S.nonces) := S.msg;
S: (S.A,S.Na,S.Nb) := d[S.nonces]S.kbs;
S:=>A(<e[<S.dst,S.kab,S.Na,S.Nb>]S.kas,e[<S.A,S.kab>]S.kbs>);
A: <-(A.msg);
A: (A.ticketa,A.ticketb) := A.msg;
A: (A.dst,A.kab,A.Na',A.Nb) := d[A.ticketa]A.kas;
A: assert(A.Na == A.Na');
A: => B(<A.ticketb,e[A.Nb]A.kab>);
B: <-(B.msg2);
B: (B.ticketb,B.nonce) := B.msg2;
B: (B.A,B.kab) := d[B.ticketb]B.kbs;
B: B.Nb' := d[B.nonce]B.kab;
B: assert(B.Nb' == B.Nb);
 *** End of Protocol,  simplified predicate follows

TRUE
```

Figure 6.22

The Yahalom protocol protects against replay attacks by including a nonce in each protocol message. Though the second component of the third message does not contain a nonce, it is protected because the key within that component is combined with a nonce in the other component of the message. Once again, by assuming the equivalence of the principal's private keys with the authentication server, the protocol meets its goals of nonce equivalence.

Syverson gives a two way authentication protocol (Figure 6.23) and an attack on the protocol (Figure 6.24) in [SYV93a].

Syverson Protocol

```
S: => A(S.kas);
A: <-(A.kas);
S: => B(S.kbs);
B: <-(B.kbs);
A: => B(<A.A,A.Na>);
B: <-(B.msg);
B: (B.A,B.Na) := B.msg;
B: => S(<B.B,B.Nb,e[<B.Nb,B.Na,B.A>]B.kbs>);
S: <-(S.msg9);
S: (S.B,S.Nb,S.msg1) := S.msg9;
S: (S.N'b,S.Na,S.A) := d[S.msg1]S.kbs;
S: assert(S.N'b == S.Nb);
S: => A(<S.S,S.B,S.Ns,e[<S.Na,S.Ns,S.B>]S.kas>);
A: <-(A.msgx);
A: (A.ASID,A.B,A.Ns,A.msg2) := A.msgx;
A: (A.N'a,A.N's,A.B') := d[A.msg2]A.kas;
A: assert(A.Na == A.N'a);
A: => S(<A.A,e[<A.kab,A.Ns,A.B>]A.kas>);
S: <-(S.msg7);
S: (S.A',S.msg3) := S.msg7;
S: (S.kab,S.N's,S.B') := d[S.msg3]S.kas;
S: assert(S.Ns == S.N's);
S: => B(<S.S,e[<S.Nb,S.kab,S.A>]S.kbs>);
B: <-(B.msg8);
B: (B.ASID,B.msg4) := B.msg8;
B: (B.N'b,B.kab,B.A') := d[B.msg4]B.kbs;
B: assert(B.Nb == B.N'b);
B: => A(e[B.f(B.Na)]B.kab);
A: <-(A.msg5);
A: A.N'a := d[A.msg5]A.kab;
A: assert(A.f(A.Na) == A.N'a);
X: gassert(B.kab == A.kab);
*** End of Protocol ***
****** Simplified predicate follows.

        (A.f(A.Na) == B.f(A.Na))
```

Figure 6.23

Syverson's protocol is slightly more complex than previously discussed protocols. It requires six transmissions with three to five components in each transmission. The strength of the authentication is increased by applying a function to principal A's original nonce, resulting in the final form of the verification condition shown in Figure 6.23.

The attack Syverson constructs on his own protocol exploits the similarity of message components. By extracting an encrypted component of principal A's first message in the attack session, and combining that component with information passed in the clear in the original session, the intruder is able to construct a message that meets the form of the protocol and spoofs both the authentication server and principal B into believing the intruder is actually principal A. The CPAL-ES evaluation of this attack reflects the success of the attack as long as A and B apply the same function f to nonce Na.

Attack On Syverson's Protocol

```
S: => A(S.kas);
A: <-(A.kas);
S: => B(S.kbs);
B: <-(B.kbs);
I: => B(<I.A,I.Na>);
B: <-(B.msga);
B: (B.A,B.Na) := B.msga;
B: B.msg1 := e[<B.Nb,B.Na,B.A>]B.kbs;
B: => S(<B.B,B.Nb,B.msg1>);
S: <-(S.msga);
S: (S.B,S.Nb,S.msg1) := S.msga;
S: (S.N'b,S.Na,S.A) := d[S.msg1]S.kbs;
S: assert((S.N'b == S.Nb));
S: S.msg2 := e[<S.Na,S.Ns,S.B>]S.kas;
S: -> A(<S.S,S.B,S.Ns,S.msg2>);
I: <-(I.msgc);
I: (I.S,I.B,I.Ns,I.msg2) := I.msgc;
I: => A(<I.B,I.Ns>);
A: <-(A.msga);
A: (A.A,A.Ns) := A.msga;
A: A.msg3 := e[<A.Na,A.Ns,A.A>]A.kas;
A: -> S(<A.B,A.Na,A.msg3>);
I: <-(I.msgd);
I: (I.B,I.N'a,I.msg3) := I.msgd;
I: => S(<I.A,I.msg3>);
S: <-(S.msgb);
S: (S.A',S.msg3) := S.msgb;
S: (S.kab,S.N's,S.B') := d[S.msg3]S.kas;
S: assert((S.Ns == S.N's));
S: S.msg4 := e[<S.Nb,S.kab,S.A>]S.kbs;
S: => B(<S.S,S.msg4>);
B: <-(B.msge);
B: (B.ASID,B.msg4) := B.msge;
B: (B.N'b,B.kab,B.A') := d[B.msg4]B.kbs;
B: assert((B.Nb == B.N'b));
B: B.msg5 := e[B.f(B.Na)]B.kab;
B: -> A(B.msg5);
I: <-(I.msg5);

*** End of Protocol ***
****** Simplified predicate follows.

TRUE
```

Figure 6.24

We present the Neuman and Stubblebine protocol from [NS93] in Figure 6.25 and its attack formulated by Syverson in [SYV93b] in Figure 6.26. The protocol is a symmetric key protocol that utilizes nonces to ensure message freshness. The protocol is structured so that several message components are forwarded between participants, complicating the protocol and making the tracking of the components more difficult. It is this nesting that [at least indirectly] provides the weakness that Syverson exploits in constructing his attack on this protocol.

CPAL-ES handles the nesting of message components easily and displays the necessary relationships automatically. The nesting results in a very complex initial verification condition. However, CPAL-ES simplification reduces the final result into manageable form.

Neuman and Stubblebine Protocol

```
S: => A(S.kas);
A: <-(A.kas);
S: => B(S.kbs);
B: <-(B.kbs);
A: => B(<A.A,A.Na>);
B: <-(B.msg1);
B: (B.src,B.Na) := B.msg1;
B: => S(<B.B,e[<B.src,B.Na,B.Tb>]B.kbs,B.Nb>);
S: <-(S.msg2);
S: (S.B,S.msg3,S.Nb) := S.msg2;
S: (S.src,S.Na,S.Tb) := d[S.msg3]S.kbs;
S: => A(<e[<S.B,S.Na,S.kab,S.Tb>]S.kas,e[<S.src,S.kab,S.Tb>]S.kbs,S.Nb>);
A: <-(A.msg4);
A: (A.msg5,A.msg6,A.Nb) := A.msg4;
A: (A.B,A.N'a,A.kab,A.Tb) := d[A.msg5]A.kas;
A: assert((A.Na == A.N'a));
A: => B(<A.msg6,e[A.Nb]A.kab>);
B: <-(B.msg9);
B: (B.msg7,B.msg8) := B.msg9;
B: (B.src',B.kab,B.Tb) := d[B.msg7]B.kbs;
B: assert(B.src == B.src');
B: B.N'b := d[B.msg8]B.kab;
B: assert(B.Nb == B.N'b);

 *** End of Protocol ***
******Original verification condition omitted due to size
****** Simplified predicate follows.

TRUE
```

Figure 6.25

Syverson constructs his attack on Newman and Stubblebine's protocol by recognizing the similarity of a message component of the first message of the target principal (B) and a message component of the authentication server (S). In the B's first message, one component contains the encryption of the catenation of the source identifier, a nonce (Nb), and a timestamp generated locally (Tb), all encrypted under S's private key shared with B (kbs). In the message from S to the originator of the protocol, one component contains the encryption of the catenation of the source identifier, the new session key, and Tb, again encrypted under kbs. Because B cannot distinguish a nonce from a key, Syverson is

able to use the message component B generated to spoof B into using the nonce as the session private key. Since the value of Na is public, the intruder can conduct a session with B, masquerading as A.

The CPAL-ES evaluation clearly reflects the intruder's actions in this attack. Messages are intercepted and the critical component for the attack is positioned to meet the protocol goal. The result of the evaluation reflects the substitution that Syverson concocted.

Attack on the Neuman and Stubblebine Protocol

```
S: => B(S.kbs);
B: <-(B.kbs);
I: => B(<I.A,I.Na>);
B: <-(B.msg1);
B: (B.A,B.Na) := B.msg1;
B: -> S(<B.B,e[<B.A,B.Na,B.Tb>]B.kbs,B.Nb>);
I: <-(I.msga);
I: (I.B,I.msgb,I.Nb) := I.msga;
I: => B(<I.msgb,e[I.Nb]I.Na>);
B: <-(B.msgc);
B: (B.msg2,B.msg3) := B.msgc;
B: (B.A,B.kab,B.Tb) := d[B.msg2]B.kbs;
B: B.N'b := d[B.msg3]B.kab;
B: if ((B.Nb == B.N'b))  then
{assert(B.good(B.kab,B.A));}
  else
{assert(B.bad(B.kab));}

 *** End of Protocol ***
****** Simplified predicate follows.

B.good(I.Na,I.A)
```

Figure 6.26

In [BIRD93], the authors give two protocols that utilize an xor function to help ensure the freshness of a message. The simple protocol from their Figure 8 is given along with its CPAL-ES evaluation in Figure 6.27. We use the CPAL function operation to reflect application of XOR to the nonce and the identifier in the second message. The

result shows that the assumption of the equivalence of the private symmetric key between A and B is not quite strong enough to complete the proof. The evaluation also shows that we must assume that A and B are applying the same function for the protocol to operate correctly.

Bird's XOR Protocol

```
A: => B(A.kab);
B: <-(B.kab);
A: => B(A.Na);
B: <-(B.Na);
B: => A(<e[B.XOR(B.Na,B.B)]B.kab,e[B.Nb]B.kab>);
A: <-(A.msg);
A: (A.msg1,A.msg2) := A.msg;
A: A.xor_Na_b := d[A.msg1]A.kab;
A: A.tmp := A.XOR(A.Na,A.B);
A: assert(A.xor_Na_b == A.tmp);
A: A.Nb := d[A.msg2]A.kab;
A: => B(A.Nb);
B: <-(B.Nb');
B: assert(B.Nb' == B.Nb);
 *** End of Protocol ***

((TRUE and (d[<e[B.XOR(A.Na,B.B)]A.kab,e[B.Nb]B.kab>.2]A.kab == B.Nb)) and
(d[<e[B.XOR(A.Na,B.B)]A.kab,e[B.Nb]B.kab>.1]A.kab == A.XOR(A.Na,A.B)))
****** Simplified predicate follows.

(B.XOR(A.Na,B.B) == A.XOR(A.Na,A.B))
```

Figure 6.27

Bird et. al. also give an attack on the XOR protocol. We give the attack and its CPAL-ES evaluation in Figure 6.28. The attack is based on a parallel session, or inter-leaving attack. With the attack on Bird's XOR protocol, we again illustrate how CPAL-ES can be used to investigate intruder actions relative to a specific protocol.

Attack on Bird's XOR Protocol

```
A: => B(A.kab);    B: <-(B.kab);
A: A.N1 := new;
A: -> B(A.N1);
I: <-(I.N1);
I: => A(I.XOR(I.A,I.XOR(I.N1,I.B)));
A: <-(A.N1');
A: A.N2 := new;
A: -> B(<e[A.XOR(A.N1',A.B)]A.kab,e[A.N2]A.kab>);
I: <-(I.msg2);
I: => A(I.msg2);
A: <-(A.msg2);
A: (A.msg3,A.msg4) := A.msg2;
A: A.xor_n1_b := d[A.msg3]A.kab;
A: A.tmp := A.XOR(A.N1,A.B);
A: assert((A.xor_n1_b == A.tmp));
A: A.N2 := d[A.msg4]A.kab;
A: -> B(A.N2);
I: <-(I.N2);

 *** End of Protocol, simplified predicate follows.

(A.XOR(I.XOR(I.A,I.XOR(unique.v1,I.B)),A.B) == A.XOR(unique.v1,A.B))
```

Figure 6.28

Chapter 7
CPAL-ES and BAN Logic

As we discussed in Chapter 5 and illustrated in Chapter 6, CPAL-ES offers message based evaluation of cryptographic protocols. This evaluation focuses on the relationships between values passed in the protocol messages and on the state of the principal's address spaces during and after completion of the protocol run. As Burrows, Abadi, and Needham so powerfully illuminated, a greater abstraction to allow the analyst to deal with concepts such as beliefs of principals is helpful, maybe necessary, to effectively evaluate the capabilities of a protocol. Seemingly, it would be beneficial if we could combine CPAL-ES evaluation with BAN Logic evaluation to yield a more powerful tool. CPAL was designed with this capability in mind.

7.1 Combining CPAL and BAN Logic Notation

Recall that BAN Logic is a combination of formula notation (the logical language) and rules of inference. The analyst expresses the assumptions and logical meaning of the protocol steps in the BAN notation and then utilizes the rules of inference to derive logical formulae (theorems) representing the meaning of the protocol. While BAN notation is highly symbolic for ease of use, the BAN concepts are easily captured in the more familiar predicate notation. Because a goal of CPAL is to allow the protocol analyst to encode the logical meaning of the statements into the operational protocol itself, CPAL was designed to allow logical predicates to be expressed in CPAL ASSUME and ASSERT statements.

In combining the CPAL-ES and BAN Logic techniques, the actions of the principals in the protocol are expressed using the CPAL operators for send, receive, encrypt, decrypt, etc. The protocol assumptions, expressed as BAN Logic predicates or as operations on values, are included as CPAL ASSUME statements. Protocol goals are captured in CPAL ASSERT statements. Thus, the first step in preparing a protocol for combined

CPAL-ES and BAN Logic evaluation is to encode the protocol assumptions, steps, and goals as though only the CPAL-ES, message based, evaluation were to be accomplished. The second step is to determine the valid BAN Logic assumptions for the protocol. The assumptions are encoded in CPAL ASSUME statements using predicate notation for each BAN Logic construct.

Next, the protocol goals are encoded as CPAL ASSERT statement also using predicate notation for each BAN Logic construct. Finally, the protocol steps are analyzed for BAN Logic annotations. The resulting annotations are treated as assumptions and are encoded in CPAL ASSUME statements.

One goal of CPAL-ES is to reduce the complexity of translating protocol actions into a logical formula. The essence of CPAL-ES is to take a procedural specification of a protocol an translate it into a logical definition, which we demonstrated in Chapter 6. When CPAL-ES is combined with BAN Logic, this job becomes more difficult.

The source of the difficulty is the nature of the BAN Logic belief constructs (predicates). Of the BAN Logic constructs we described in Chapter 2, only the SEES, FRESH, and GOODKEY predicates are of interest to us as beliefs to be associated with particular values or actions. Other predicates, such as the CONTROLS predicate, are used only for expressing assumptions that are not based on a principal's actions. A principal either controls a predicate or it does not. Control is not gained or transferred during a protocol run. For this type of predicate, we do not need to consider translation of principal actions into associated beliefs. Other BAN Logic predicates are concerned with the translation we address, but are not applicable for the simple examples we have chosen.

First consider the SEES predicate. Based on the CPAL operational model, we define the SEES predicate to mean that a principal SEES every data item that exists in their local address space. We can state this translation as a formal rule:

For every CPAL receive and assignment statement of the form:

"P: X:=Y;" or "P: <-(X);"

generate an assume statement of the form:

"P: assume(sees(P,X));".

Thus, CPAL receive and assignment statements are translated directly into BAN Logic SEES predicates. Additional items that a principal sees are identified during the weakest precondition evaluation and reflect the construction of the messages the principal receives. Consider the following CPAL segment:

1. A: =>B(<Na,Na1>);
2. B: <-(msg);
3. B: assume(X.sees(B.B,B.msg));

The assume statement in step 3 is manually generated directly from the receive statement in step 2 as we just described. Since msg will be received into B's local address space as B.msg, principal B is assumed to SEE msg in the BAN Logic sense. Due to the structure of the value "msg", when the weakest precondition definitions are applied to the specification, this assumption will result in the following entry in the verification condition:

sees(B.B,<A.Na,A.Na1>).

Similarly, we may define an assumption generation rule for the CPAL NEW statement. Any value generated as NEW is created in the current run of the protocol and is considered to be a random number, and, thus, is fresh by definition. We further define this statement to mean that the creator of the NEW value believes that the value is fresh. We may then state:

For every CPAL assignment statement of the form "P: X:=new;" generate two assume statements of the form

"P: assume(fresh(X));".

"P: assume(believes(P,fresh(X)));".

We now see how CPAL-ES eliminates the idealization process from BAN Logic evaluation for the standard meaning of the SEES and FRESH constructs. We point out, however, that the above rules may not be all inclusive. It may be that a particular protocol makes other assumptions about newly generated values, e.g. that the value will always be an odd number. We cannot hope to predict all possible intended meanings that protocol analysts will desire for these constructs, and do not preclude use of other assume statements to verbalize these assumptions. However, the given definitions for SEES and FRESH are sufficient for the types of protocols we have seen.

The GOODKEY predicate is essential to the BAN Logic evaluation process, and its translation is much more complex than that of the other two predicates. Again, CPAL-ES is concerned with the values that principals send as messages. On the other hand, BAN Logic is concerned with what the belief of the principal is when they send values as messages. In the case of the SEES and FRESH predicates, we can mechanically extract the meaning of the values transmitted because the model captures the essence of the predicates. The GOODKEY predicate is not captured as exactingly.

The difficulty arises because of the way the GOODKEY belief is acquired by a principal. In BAN Logic, there are no rules that allow one to arrive specifically at the belief: GOODKEY(K). GOODKEY is rather treated as an arbitrary belief. A canonical BAN Logic proof tree for an arbitrary belief is given in Appendix D. As shown in the appendix, arbitrary beliefs can only be proven in BAN Logic by beginning with the jurisdiction rule, which says that principal P can believe K is a GOODKEY if someone P trusts to believe it actually believes it. We must prove that the trusted principal said GOODKEY(K) in the present run of the protocol. This step is accomplished by exercising the BAN Logic nonce verification rule.

With the first two steps accomplished, we are unable to proceed further given only the protocol steps and the assumptions described so far. In the first two steps, reasoning was not based on principal actions, but rather only on the assumptions and goals expressed before any protocol steps occurred. In the third step, we must prove that P sees GOOD-KEY(K) encrypted under the common key shared between P and the trusted party. The SEES predicate is generated from a principal's action, i.e. a CPAL receive or send statement. The root of the problem is that CPAL send and receive statements only express the transmission and assignment of values, while what we need to SEE is a predicate (GOOD-KEY(k)).

This fact represents the primary theme of the protocol, which becomes clear when we state the central question as: "What value will the trusted party say when it intends to convey the belief GOODKEY(k)?" Consider the example in Figure 7.2. Notice the assume statement:

```
A: assume((believes(A.A(said(A.S,S.kab)) IMPLIES
    believes(A.A,said(A.S,goodkey(A.A,S.kab,A.B)))));
```

This says that when principal A determines that principal S (the authentication server) "said" the message containing the value S.kab, S actually meant that kab is a good key to be used for communication between A and B. This assumption is sufficient to complete the proof automatically. A proof tree for a BAN Logic proof is provided in Appendix D.

To illustrate how interaction between CPAL-ES and BAN Logic occurs, we provide a simple protocol, encoded in CPAL with BAN Logic goals and assumptions in Figure 7.1. As we described above, the BAN Logic assumptions and annotations are expressed as ASSUME statements in CPAL. The assumptions precede the first step of the protocol and reflect the facts the author expect to be true before the protocol run begins. Assumptions may express facts involving data across address spaces. For example, the

assumption in statement 4 indicates that principal B believes the value kab in principal A's address space is a good key between A and B.

CPAL-ES and BAN Logic Evaluation Example

1	A: =>B(k);
2	B: <-(k);
3	A: assume(X.believes(A.A,X.goodkey(A.A,A.kab,A.B)));
4	B: assume(X.believes(B.B,X.controls(B.A(X.goodkey(B.A,A.kab,B.B)))));
5	B: assume(X.believes(B.B,X.fresh(B.Nb)));
6	B: => A(e[<B.B,B.Nb>]B.k);
7	A: <-(A.msg1);
8	A: (A.B,A.Nb) := d[A.msg1]A.k;
9	A: assume(X.believes(A.A,X.fresh(A.Na)));
10	A: => B(e[<A.Na,A.kab>]A.k);
11	B: <-(B.msg);
12	B: assume(X.sees(B.B,B.msg));
13	B: (B.Na,B.kab) := d[B.msg]B.k;
14	B: assume(X.sees(B.B,X.e(X.cat(B.Na,B.kab),B.k)));
15	B: => A(e[B.Na]B.kab);
16	A: <-(A.msg2);
17	A: assume(X.sees(A.A,A.msg2));
18	A: A.N'a := d[A.msg2]A.kab;
19	A: assume(X.sees(A.A,A.N'a));
20	A: assume(X.sees(A.A,X.e(A.N'a,A.kab)));
21	A: assert(A.Na == A.N'a)
22	A: assert(A.believes(A.A,A.believes(A.B,A.goodkey(A.A,A.kab,A.B))));
	Figure 7.1

The difference between CPAL operators and BAN Logic conventions is highlighted by the use of the encryption operation in the example in Figure 7.1. In statement

10, the CPAL encryption operator is utilized to indicate that principal A is performing a computation on value "<Na,kab>". In statement 15, the ASSUME statement is expressing the fact that principal B can see the value that resulted from encrypting the catenation of the values Na and kab.Protocol goals are expressed as CPAL ASSERT statements. The logical goals described by Burrows, Abadi, and Needham [BAN89] are given as predicates. In Figure 7.1, statement 21 is the message based goal, and statement 22 is the BAN Logic goal of the protocol.

7.2 Completing the BAN Logic Proof

While predicates of any form may be expressed as goals of protocols, CPAL-ES is unable to reduce predicates other than the comparisons and simple logical reductions described in Chapter 5. The evaluation shown in Figure 7.2 gives the formal definition of the Needham and Schroeder Private Key Protocol with goals and assumptions expressed in BAN Logic notation. In this case, the message-based goals evaluate to:

(B.f(B.Nb) == A.f(B.Nb))

as was described in the CPAL evaluation of this protocol in Figure 6.5. The assert statements representing the BAN Logic goals form the conjunction of the final verification condition which is a definition of the protocol.

CPAL-ES does not recognize or understand the BAN Logic theorems or rules of inference and cannot reduce the final verification further. The proof of the final verification condition expressed as BAN Logic statements could be accomplished by encoding the BAN rules of inference into CPAL-ES. However, there are many other logics that also could be combined with CPAL-ES to extend the power of each mechanism. Fortunately, many people devote their main research to developing theorem proving techniques and mechanical theorem proof systems. There are several such systems available for accomplishing this final proof. Boyer and Moore [BM79] and [BM88] developed what is now

the ad hoc standard theorem proof system known appropriately as the Boyer/Moore Theorem Prover.

The power and effectiveness of the Boyer/Moore Theorem Prover notwithstanding, we chose to utilize a simpler proof system, developed by Owre, Shankar, and Rushby [OSR93a], [OSR93b] to prove final verification conditions produced by CPAL-ES. The proof system the authors call "The Prototype Verification System" or PVS, mechanizes proof steps such as the application of a rule of inference. PVS is thoroughly documented with tutorials, reference manuals, and guides for use and is available in the public domain.

To capture the BAN Logic rules of inference in the PVS language, several notational adjustments were necessary. As with the CPAL representation of BAN Logic constructs, the constructs were changed to predicate notation for PVS. BAN Logic predicates (SEES, GOODKEY, etc.) were declared as boolean functions and the BAN Logic rules were encoded as PVS axioms.

The PVS representation of BAN Logic is given in Figure 7.3. There are twelve rules in the description. In most cases, the rule name reflects the BAN Logic formula they represent, i.e. SEES, MSG_MEANING, etc. The declarations are shown in two parts, the first being the function identifiers that are needed in the BAN Logic rules definitions themselves. The second part contains the declarations of identifiers that are expected to be used in the PVS representation of the protocol, i. e. the verification condition. The declarations shown in Figure 7.3 are the identifiers used for evaluation of the Needham and Schroeder Private Key protocol, discussed and given later in this chapter.

CPAL-ES BAN Logic Evaluation of the Needham and Schroeder Protocol

```
S: => A(S.kas);  A: <-(A.kas);    S: => B(S.kbs);   B: <-(B.kbs);
A: assume(believes(A.A,goodkey(A.A,S.kas,A.S)));
B: assume(believes(B.B,goodkey(B.B,S.kbs,B.S)));
A: assume(believes(A.A,controls(A.S,goodkey(A.A,S.kab,A.B))));
B: assume(believes(B.B,controls(B.S,goodkey(B.A,S.kab,B.B))));
A: assume(believes(A.A,fresh(A.Na)));
B: assume(believes(B.B,fresh(B.Nb)));
B: assume(believes(B.B,fresh(B.A)));
A: assume(believes(A.A,said(A.S,goodkey(A.A,S.kab,A.B))));
A: assume((believes(A.A,said(A.S,S.kab)) IMPLIES believes(A.A,said(A.S,goodkey(A.A,S.kab,A.B)))));
B: assume((believes(B.B,said(B.S,S.kab)) IMPLIES believes(B.B,said(B.S,goodkey(B.A,S.kab,B.B)))));
A: assume((believes(A.A,said(A.B,A.Nb)) IMPLIES believes(A.A,said(A.B,goodkey(A.A,S.kab,A.B)))));
B: assume((believes(B.B,said(B.A,B.f(B.Nb))) IMPLIES believes(B.B,said(B.A,good-
key(B.A,S.kab,B.B)))));
A: => S(e[<A.A,A.B,A.Na>]A.kas);   S: <-(S.msg);
S: (S.A,S.B,S.Na) := d[S.msg]S.kas;
S: => A(e[<S.Na,S.B,S.kab,e[<S.kab,S.A>]S.kbs>]S.kas);   A: <-(A.msg3);
A: assume(sees(A.A,A.msg3));
A: (A.N'a,A.B',A.kab,A.ticket) := d[A.msg3]A.kas;
A: assume(sees(A.A,cat(A.N'a,A.kab)));
A: assume(sees(A.A,A.ticket));
A: assert((A.Na == A.N'a));
A: => B(A.ticket);     B: <-(B.ticket);
B: assume(sees(B.B,B.ticket));
B: (B.kab,B.A') := d[B.ticket]B.kbs;
B: assume(sees(B.B,cat(B.A,B.kab)));
B: => A(e[B.Nb]B.kab);    A: <-(A.msg4);
A: assume(sees(A.A,A.msg4));
A: A.Nb := d[A.msg4]A.kab;
A: assume(sees(A.A,A.Nb));
A: assert(A.believes(A.A,A.believes(A.B,A.goodkey(A.A,A.kab,A.B))));
A: => B(e[A.f(A.Nb)]A.kab);   B: <-(B.msg5);
B: assume(sees(B.B,B.msg5));
B: B.N'b := d[B.msg5]B.kab;
B: assert((B.f(B.Nb) == B.N'b));
B: assert(B.believes(B.B,B.believes(B.A,B.goodkey(B.A,B.kab,B.B))));
 ** End of Protocol.  Simplified predicate follows. **
((((((((((((((((((((B.believes(B.B,B.believes(B.A,B.goodkey(B.A,S.kab,B.B))) and (B.f(B.Nb) ==
A.f(B.Nb))) or not (sees(B.B,e[A.f(B.Nb)]S.kab))) and A.believes(A.A,A.believes(A.B,A.good-
key(A.A,S.kab,A.B)))) or not (sees(A.A,B.Nb))) or not (sees(A.A,e[B.Nb]S.kab))) or not
(sees(B.B,cat(B.A,S.kab)))) or not (sees(B.B,e[<S.kab,A.A>]S.kbs))) or not
(sees(A.A,e[<S.kab,A.A>]S.kbs))) or not (sees(A.A,cat(A.Na,S.kab)))) or not
(sees(A.A,e[<A.Na,A.B,S.kab,e[<S.kab,A.A>]S.kbs>]S.kas))) or not ((not
(believes(B.B,said(B.A,B.f(B.Nb))))or believes(B.B,said(B.A,goodkey(B.A,S.kab,B.B)))))) or not ((not
(believes(A.A,said(A.B,A.Nb))) or believes(A.A,said(A.B,goodkey(A.A,S.kab,A.B)))))) or not ((not
(believes(B.B,said(B.S,S.kab))) or believes(B.B,said(B.S,goodkey(B.A,S.kab,B.B)))))) or not ((not
(believes(A.A,said(A.S,S.kab))) or believes(A.A,said(A.S,goodkey(A.A,S.kab,A.B))))))or not
(believes(A.A,said(A.S,goodkey(A.A,S.kab,A.B))))) or not (believes(B.B,fresh(B.A)))) or not
(believes(B.B,fresh(B.Nb)))) or not (believes(A.A,fresh(A.Na))))or not (believes(B.B,controls(B.S,good-
key(B.A,S.kab,B.B))))) or not (believes(A.A,controls(A.S,goodkey(A.A,S.kab,A.B))))) or not
(believes(B.B,goodkey(B.B,S.kbs,B.S)))) or not (believes(A.A,goodkey(A.A,S.kas,A.S))))
```

Figure 7.2

We found that while BAN Logic contains sufficient power to allow evaluation of protocols manually, addition of a few simple AXIOMS greatly simplifies formal application. Mostly, these supplemental rules allow easy extracting of components of an encrypted message (SEES_ENC) and reordering of identifiers in predicates whose arguments are symmetrical (BELIEVES_SYM, etc.). The supplemental rules we used are shown in Figure 7.4.

PVS Entries for BAN Logic Rules

```
BAN       : THEORY   BEGIN
e,d,ep,dp,signature,cat:                                    PRED[[bool,bool]]
believes ,sees,said,controls,fresh,goodkey,pubkey,secret:   PRED[[bool,bool]]
A,B,S,S_kas,S_kbs,S_A,S_B,A_kab,B_kab,B_A,B_B,S_S,B_S,B_kbs:bool
A_Na,A_Nc,B_Nb,B_Nc,A_A,A_B,S_kab,A_S,A_kas:                bool
B_Nb1,B_Nc1, A_Na1,B_msg4,A_msg,B_msg2,B_src:               bool
 % Ban Rules of inference
msg_meaning1:  AXIOM FORALL (P,Q,KPQ,X: bool):
    believes(P,(goodkey(P,KPQ,Q))) AND sees(P,e(X,KPQ)) IMPLIES
          believes(P,said(Q,X))
msg_meaning2:  AXIOM  FORALL (P,Q,PUB_Q,X:bool): %     For  public keys
    believes(P,pubkey(Q,PUB_Q)) and sees(P,e(X,PUB_Q))  IMPLIES
          believes(P,said(Q,X))
msg_meaning3:  AXIOM  FORALL (P,Q,Y,X:bool):    %     For shared  secrets
    believes(P,secret(Q,Y,P)) and sees(P,signature(X,Y)) IMPLIES
          believes(P,said(Q,X))
nonce_verification:  AXIOM  FORALL (P,Q,X: bool):
    believes(P,fresh(X)) and believes(P,said(Q,X)) IMPLIES
          believes(P,believes(Q,X))
jurisdiction:  AXIOM  FORALL (P,Q,X: bool):
    believes(P,controls(Q,X)) and believes(P,believes(Q,X))  IMPLIES believes(P,X)
sees1: AXIOM  FORALL (P,X,Y: bool): sees(P,cat(X,Y)) IMPLIES
    sees(P,X) and sees(P,Y)
sees2: AXIOM FORALL (P,X,Y: bool): sees(P,signature(X,Y))  IMPLIES sees(P,X)
sees3: AXIOM FORALL (P,Q,PUB_P,X: bool):
    believes(P,pubkey(P,PUB_P)) and sees(P,e(X,PUB_P))  IMPLIES  sees(P,X)
sees4: AXIOM FORALL (P,Q,X,KPQ: bool):
    believes(P,goodkey(P,KPQ,Q)) and sees(P,e(X,KPQ)) IMPLIES sees(P,X)
sees5: AXIOM FORALL (P,Q,PUB_Q,PUB_Q_INV,X: bool):
    believes(P,pubkey(Q,PUB_Q)) and sees(P,e(X,PUB_Q_INV))  IMPLIES
          sees(P,X)
freshness1:  AXIOM FORALL (P,X,Y: bool):
    believes(P,fresh(X)) AND sees(P,cat(X,Y)) IMPLIES believes(P,fresh(cat(X,Y)))
```

Figure 7.3

Some notational adjustments are necessary to merge BAN and PVS representations of the logical notions. For example, we forgo the BAN symbolic notation in lieu of predicate notation for standard concepts such as “goodkey” and “sees”.

PVS Supplement for BAN Logic Rules

```
freshness2b: AXIOM FORALL (P,X,Y: bool):
    believes(P,fresh(cat(X,Y))) IMPLIES believes(P,fresh(Y))
freshness1b: AXIOM FORALL (P,X,Y: bool):
    believes(P,fresh(Y)) IMPLIES believes(P,fresh(cat(X,Y)))
freshness2:  AXIOM     FORALL (P, X, Y: bool):
    believes(P, fresh(cat(X, Y))) IMPLIES believes(P, fresh(Y)) AND
    believes(P, fresh(X))
freshness3: AXIOM FORALL (P,Q,kpq: bool):
    believes(P,fresh(kpq)) IMPLIES
            believes(P,fresh(goodkey(P,kpq,Q))) AND
            believes(P,fresh(goodkey(Q,kpq,P)))
sees_enc: AXIOM  FORALL (P,X,Y,K: bool):
    sees(P,e(cat(X,Y),K)) IMPLIES sees(P,e(X,K)) and sees(P,e(Y,K))
believes_sym: AXIOM  FORALL (P,Q,k: bool):
    believes(P,goodkey(Q,k,P)) IMPLIES believes(P,goodkey(P,k,Q))
said_sym:  AXIOM FORALL (P,Q,R,T,k:bool):
    believes(P,said(Q,goodkey(R,k,T))) IMPLIES believes(P,said(Q,goodkey(T,k,R)))
controls_sym: AXIOM  FORALL (P,Q,R,k: bool):
    believes(P,controls(Q,goodkey(P,k,R))) IMPLIES
            believes(P,controls(Q,goodkey(R,k,P)))
```

Figure 7.4

The above description addresses translation of the BAN Logic constructs and rules of inference into PVS notation. We now turn our attention into utilizing the output from CPAL-ES in PVS where several additional adjustments are necessary. Because the dot notation used in CPAL to reflect an identifier prefix has a different meaning in PVS, we replace the periods in the CPAL identifier names with underscores. We must also replace the angle brackets which CPAL recognizes as the catenation operator with the predicate cat(). Similarly, the encryption operator is translated from e[X]k to its predicate equivalent e(X,k).

Another adjustment involves the prefix to BAN Logic predicate identifiers.

CPAL forces ***every*** identifier to be coupled to an acting principal. Since we routinely assume that every principal's version of the BAN Logic predicates are the same, we utilize principal X as the acting ID prefix for the BAN Logic predicates. Since these predicates are globally identical, we can omit the prefix for these identifiers in the PVS version of the protocol.

The final PVS file used to complete the proof of the verification condition is attained by combining the PVS representation of the BAN Logic rules given in Figure 7.3 and 7.4, with the translated verification condition produced by CPAL-ES. As we describe above, there are several steps necessary to prepare the input for PVS from CPAL-ES combined with BAN Logic. We list the necessary steps in Figure 7.5. The axioms and conjecture for the Needham and Schroeder protocol are given in Figure 7.6.

Steps For Creating The CPAL-ES/BAN Logic/ PVS File

1. Encode the protocol steps, assumptions, and goals in CPAL
2. Generate the appropriate assume statements from the receive, assignment and "new" statements in the specification.
3. Execute CPAL-ES against the specification
4. Extract the simplified verification from the file "protocol.out"
5. Find/replace all occurrences of:

"."	to	"_"
"e["	to	"e("
"]keyid"	to	",keyid)"
"<"	to	"cat("
">"	to	")"
"X."	to	""

6. Delete all remaining prefixes to BAN Logic predicates.
7. Scan the verification condition to ensure there are no more than two arguments for any "cat" operator.
8. Combine the PVS file for BAN Logic with the verification condition.
9. Add any necessary declarations for identifiers in the verification condition.
10. Construct the CONJECTURE declaration for the verification condition.

Figure 7.5

In PVS, the proof is completed by an analyst interacting with the system to derive a set of steps that prove the theorem or theorems that represent the protocol goals. The

PVS commands to accomplish these proofs are documented in [OSR93a], [OSR93b], [SOR93b] and illustrated in [SOR93a], and [CORSS95]. The technique required for the two proofs we accomplished involves simple deductive reasoning.

Utilizing the conjecture statement in the proof file, PVS automatically produces a sub-goal to be proven, in our case, representing the message-based and BAN Logic goals for the protocol. We first analyze the sub-goal to identify an axiom we may apply to move closer to a final proof of the conjecture. The axiom is applied by use of the PVS LEMMA rule. For example, if we have the following sub-goal:

believes(A_A,goodkey(A_A,S_kab,A_B))

we may elect to apply the jurisdiction axiom:

jurisdiction: AXIOM FORALL (P,Q,X: bool):
believes(P,controls(Q,X)) and believes(P,believes(Q,X)) IMPLIES believes(P,X)

by invoking the LEMMA rule as:

(lemma "jurisdiction" ("P" "A_A" "Q" "A_S" "X" "goodkey(A_A,S_kab,A_B)"))

The LEMMA rule tells PVS to apply the jurisdiction axiom, replacing the universal variables P and Q with the constant variables A_A and A_S, and X with the predicate "goodkey(A_A,S_kab,A_B)". The result of the application of this rule would be to generate a sub-goal requiring the proof of the antecedent of the jurisdiction rule, i.e. that:

believes(A_A,controls(A_S,goodkey(A_A,S_kab,A_B))) and
believes(A_A,believes(A_S,goodkey(A_A,S_kab,A_B)))

At this point, we would apply the PVS rule SPLIT to split the conjunction and allow us to focus on one part of this goal, either attempting to prove that A actually does believe that S controls kab, or that A believes that S believes kab is a good key. Both of

these must be proven to complete the proof.

Ultimately, the proof of a sub-goal is accomplished when a sub-goal reduces to one of the assumed predicates. In the above sequence the first sub-goal

(A_A believes A_S controls S_kab)

is identical to an assumed predicate in Figure 7.6. This sub-goal is automatically reduced by PVS.

PVS File Entries for the Needham and Schroeder Private Key Protocol

```
gnsgoal: CONJECTURE ((((((((((((((((((((
believes(B_B,believes(B_A,goodkey(B_A,S_kab,B_B)))
and (B_f(B_Nb) == A_f(B_Nb))) or not (sees(B_B,e(B_f(B_Nb),S_kab)))) and
believes(A_A,believes(A_B,goodkey(A_A,S_kab,A_B)))) or
not (sees(A_A,B_Nb))) or not (sees(A_A,e(A_Nb,S_kab)))) or
not (sees(B_B,cat(B_A,S_kab)))) or not (sees(B_B,e(cat(S_kab,B_A),S_kbs)))) or
not (sees(A_A,e(cat(S_kab,A_A),S_kbs)))) or
not (sees(A_A,cat(A_Na,S_kab)))) or not
(sees(A_A,e(cat(cat(A_Na,S_kab),e(cat(S_kab,A_A),S_kbs)),S_kas))))
or not ((not (believes(B_B,said(B_A,B_f(B_Nb)))) or
believes(B_B,said(B_A,goodkey(B_A,S_kab,B_B)))))) or
not ((not (believes(A_A,said(A_B,A_Nb))) or
believes(A_A,said(A_B,goodkey(A_A,S_kab,A_B)))))) or
not ((not (believes(B_B,said(B_S,S_kab))) or
believes(B_B,said(B_S,goodkey(B_A,S_kab,B_B)))))) or
not ((not (believes(A_A,said(A_S,S_kab))) or
believes(A_A,said(A_S,goodkey(A_A,S_kab,A_B)))))) or
not (believes(A_A,said(A_S,goodkey(A_A,S_kab,A_B))))) or
not (believes(B_B,fresh(B_A)))) or not (believes(B_B,fresh(B_Nb)))) or
not (believes(A_A,fresh(A_Na)))) or
not (believes(B_B,controls(B_S,goodkey(B_B,S_kab,B_A))))) or
not (believes(A_A,controls(A_S,goodkey(A_A,S_kab,A_B))))) or
not (believes(B_B,goodkey(B_B,S_kbs,B_S)))) or
not (believes(A_A,goodkey(A_A,S_kas,A_S))))

END gnsban
```

Figure 7.6

A sample of the PVS proof for the Needham and Schroeder protocol is given in Figure 7.7 and the complete PVS proof is given in Appendix E.

PVS Proof for the Needham and Schroeder Private Key Protocol

```
(|gnsban|
 (|gnsgoal| "" (FLATTEN)
  (("" (SPLIT +)
    (("1" (FLATTEN)
      (("1" (SPLIT +) (("1" (LEMMA "nonce_verification"
          ("P" "B_B" "Q" "B_A" "X" "goodkey(B_A,S_kab,B_B)"))
         (("1" (SPLIT -1)
           (("1" (PROPAX) NIL)
            ("2" (LEMMA "freshness3" ("P" "B_B" "Q" "B_A" "kpq" "S_kab"))
             (("2" (SPLIT -1)
               (("1" (FLATTEN) (("1" (PROPAX) NIL)))
                ("2" (LEMMA "freshness2" ("P" "B_B" "Y" "B_A" "X" "S_kab"))
                 (("2" (SPLIT -1)
                   (("1" (FLATTEN) (("1" (PROPAX) NIL))) ("2"
                    (LEMMA "freshness1b" ("P" "B_B" "Y" "B_A" "X" "S_kab"))
                    (("2" (SPLIT -1)
                      (("1" (PROPAX) NIL) ("2" (PROPAX) NIL)))))))))))))
           ("3" (SPLIT -9)  (("1"
             (LEMMA "msg_meaning1"
              ("P" "B_B" "Q" "B_A" "KPQ" "S_kab" "X" "B_f(B_Nb)"))
             (("1" (SPLIT -1)
               (("1" (PROPAX) NIL)  ("2"
                 (LEMMA "jurisdiction"
                  ("P" "B_B" "Q" "B_S" "X" "goodkey(B_B,S_kab,B_A)"))
                 (("2" (SPLIT -1)
                   (("1" (PROPAX) NIL) ("2" (PROPAX) NIL)

                  ***** The majority of the proof is omitted due to space. *****

                      (LEMMA "freshness3"
                       ("P" "A_A" "Q" "A_B" "kpq" "S_kab"))
                       (("2" (SPLIT -1)
                         (("1" (FLATTEN) (("1" (PROPAX) NIL))) ("2"
                           (LEMMA "freshness2"
                            ("P" "A_A" "X" "A_Na" "Y" "S_kab")) (("2" (SPLIT -1)
                             (("1" (FLATTEN) (("1" (PROPAX) NIL))) ("2"
                               (LEMMA "freshness1"
                                ("P" "A_A" "X" "A_Na" "Y" "S_kab")) (("2" (SPLIT -1)
                                 (("1" (PROPAX) NIL) ("2" (PROPAX) NIL)))))))))))
                      ("3" (PROPAX) NIL)))))))))
          ("3" (POSTPONE) NIL)))))
       ("2" (POSTPONE) NIL)))))))))))
```

Figure 7.7

For emphasis and illustration we also evaluated Snekkenes KP protocol from [SNEK92] using CPAL-ES, BAN Logic and PVS. The CPAL-ES evaluation of the KP protocol without BAN Logic constructs was given in Figure 6.19. The CPAL-ES definition of the protocol with BAN Logic goals and assumptions is given in Figure 7.8 and the PVS file entries for proof of the verification condition are in Figure 7.9. The PVS proof of the verification condition for the KP protocol is provided in Appendix F.

Combining CPAL-ES and BAN Logic on Snekkenes' KP Protocol

```
S: => A(S.kas); A: <-(A.kas);
S: => B(S.kbs); B: <-(B.kbs);
A: assume(X.believes(A.A,X.goodkey(A.A,S.kas,A.S)));
B: assume(X.believes(B.B,X.goodkey(B.B,S.kbs,B.S)));
A: assume(X.believes(A.A,X.fresh(A.Na)));
S: assume(X.believes(S.S,X.goodkey(S.A,S.kab,S.B)));
S: assume(X.said(S.S,X.goodkey(S.A,S.kab,S.B)));
B: assume(X.believes(B.B,X.controls(B.S,X.fresh(S.kab))));
B: assume(X.believes(B.B,X.controls(B.S,X.goodkey(B.B,S.kab,B.A))));
B: assume(X.believes(B.B,X.fresh(B.Nb)));
B: assume(X.believes(B.B,X.controls(B.S,S.kab)));
A: assume((X.believes(A.A,X.said(A.S,S.kab)) IMPLIES
        X.believes(A.A,X.said(A.S,X.goodkey(A.A,S.kab,A.B)))));
B: assume((X.believes(B.B,X.said(B.S,S.kab)) IMPLIES
        X.believes(B.B,X.said(B.S,X.goodkey(B.A,S.kab,B.B)))));
A: assume((X.believes(A.A,X.said(A.B,A.Na)) IMPLIES
        X.believes(A.A,X.said(A.B,X.goodkey(A.A,S.kab,A.B)))));
B: assume((X.believes(B.B,X.said(B.A,B.Nc)) IMPLIES
        X.believes(B.B,X.said(B.A,X.goodkey(B.A,S.kab,B.B)))));
A: assume(X.believes(A.A,X.controls(A.S,X.goodkey(A.A,S.kab,A.B))));
A: assume(X.believes(A.A,X.fresh(A.Na)));
A: assume(X.believes(A.A,X.controls(A.S,S.kab)));
B: assume(X.believes(B.B,X.fresh(B.Nc)));
A: => B(<A.A,A.Na,A.B>);
B: <-(B.msg);
B: assume(X.sees(B.B,B.msg));
B: (B.A',B.Na,B.B') := B.msg;
B: assume(X.sees(B.B,B.Na));
B: => S(<B.A',B.Na,B.B,B.Nb>);  S: <-(S.msg);
S: (S.A,S.Na,S.B,S.Nb) := S.msg;
S: => B(e[<S.kab,S.A,S.Nb,e[<S.A,S.B,S.Na,S.kab>]S.kas>]S.kbs);
B: <-(B.msg2);
B: assume(X.sees(B.B,B.msg2));
B: (B.kab,B.A",B.Nb',B.msg3) := d[B.msg2]B.kbs;
B: assert((B.A" == B.A'));
B: assume(X.sees(B.B,B.msg3));
B: assume(X.sees(B.B,X.cat(B.kab,B.Nb')));
B: assert((B.Nb == B.Nb'));
B: => A(<B.msg3,e[<B.Na,B.Nc,B.B>]B.kab>);   A: <-(A.msg);
A: assume(X.sees(A.A,A.msg));
A: (A.msg3,A.msg4) := A.msg;
A: assume(X.sees(A.A,A.msg3));
A: assume(X.sees(A.A,A.msg4));
A: (A.A',A.B',A.Na',A.kab) := d[A.msg3]A.kas;
A: assume(X.sees(A.A,X.cat(A.Na',A.kab)));
A: assert((A.Na == A.Na'));
A: (A.Na",A.Nc,A.B") := d[A.msg4]A.kab;
A: assume(X.sees(A.A,X.cat(A.Na",A.Nc)));
A: assert((A.Na == A.Na"));
A: => B(e[<A.Nc,A.A>]A.kab);  B: <-(B.msg4);
B: assume(X.sees(B.B,B.msg4));
B: (B.Nc',B.A") := d[B.msg4]B.kab;
B: assume(X.sees(B.B,X.cat(B.Nc,B.A')));
B: assert((B.Nc == B.Nc'));
A: assert(A.believes(A.A,A.believes(A.B,A.goodkey(A.A,A.kab,A.B))));
B: assert(B.believes(B.B,B.believes(B.A,B.goodkey(B.A,B.kab,B.B))));
 *** End of Protocol ***
```

Figure 7.8

The KP protocol is very similar to the Needham and Schroeder protocol. There are ten more assumptions, though the proofs of the protocols are of similar complexity.

PVS Definition of Snekkenes' KP Protocol

```
gkpgoal: CONJECTURE
((((((((((((((((((((((((((((((believes(B_B,believes(B_A,goodkey(B_A,S_kab,B_B)))
and believes(A_A,believes(A_B,goodkey(A_A,S_kab,A_B)))) or
not (sees(B_B,cat(B_Nc,A_A)))) or
not (sees(B_B,e(cat(B_Nc,A_A),S_kab)))) or
not (sees(A_A,cat(A_Na,B_Nc)))) or
not (sees(A_A,cat(A_Na,S_kab))))  or
not (sees(A_A,e(cat(A_Na,B_Nc),S_kab)))) or
not (sees(A_A,e(cat(A_Na,S_kab),S_kas)))) or
not(sees(A_A,cat(e(cat(A_Na,S_kab),S_kas),e(cat(A_Na,B_Nc),S_kab))))) or not
(sees(B_B,cat(S_kab,B_Nb))))   or
not (sees(B_B,e(cat(A_Na,S_kab),S_kas)))) or not
(sees(B_B,e(cat(cat(S_kab,A_A),cat(B_Nb,e(cat(A_Na,S_kab),S_kas))),S_kbs)))) or
not (sees(B_B,A_Na))) or
not (sees(B_B,cat(A_A,A_Na)))) or
not (believes(B_B,fresh(B_Nc)))) or
not (believes(A_A,controls(A_S,S_kab)))) or
not (believes(A_A,fresh(A_Na)))) or
not (believes(A_A,controls(A_S,goodkey(A_A,S_kab,A_B))))) or
not ((not (believes(B_B,said(B_A,B_Nc))) or
believes(B_B,said(B_A,goodkey(B_A,S_kab,B_B))))) or
not ((not (believes(A_A,said(A_B,A_Na))) or
believes(A_A,said(A_B,goodkey(A_A,S_kab,A_B))))) or
not ((not (believes(B_B,said(B_S,S_kab))) or
believes(B_B,said(B_S,goodkey(B_A,S_kab,B_B))))) or
not ((not (believes(A_A,said(A_S,S_kab))) or
believes(A_A,said(A_S,goodkey(A_A,S_kab,A_B))))) or
not (believes(B_B,controls(B_S,S_kab)))) or
not (believes(B_B,fresh(B_Nb)))) or
not (believes(B_B,controls(B_S,goodkey(B_B,S_kab,B_A))))) or
not (believes(B_B,controls(B_S,fresh(S_kab))))) or
not (said(S_S,goodkey(S_A,S_kab,S_B)))) or
not (believes(S_S,goodkey(S_A,S_kab,S_B)))) or
not (believes(A_A,fresh(A_Na)))) or
not (believes(B_B,goodkey(B_B,S_kbs,B_S)))) or
not (believes(A_A,goodkey(A_A,S_kas,A_S))))

END gkpban
```

Figure 7.9

7.3. Chapter Summary

In chapter 6 we demonstrate how CPAL-ES is utilized to atomically determine a formal definition for a wide variety of protocols. In this chapter we show how CPAL-ES can be combined with BAN Logic to provide a consolidated definition and evaluation of cryptographic protocols, reaching beyond the values passed in the protocol messages and actions of the protocol and assessing the intent of the actions. BAN Logic goals and assumptions are encoded in CPAL ASSUME and ASSERT statements to formalize the meaning of the protocol step and also of the intended meaning behind the steps and principal's actions.

We showed in this chapter how CPAL made the idealization process more formal by allowing mechanical evaluation of the BAN SEES and FRESH predicates. We illustrate these concepts through the actual evaluation of real protocols and show how the system effectively illuminates the classic Needham and Schroeder protocol flaw.

We also show how CPAL-ES can easily interact with a formal theorem proving system to evaluate assumptions and goals expressed in an epistemic logic. The same assumption mechanism CPAL-ES uses to simplify formulas based on message-based goals is used to formally specify goals and assumptions peculiar to any number of logics. CPAL-ES produces output that may be utilized as input to a mechanical proof process.

Chapter 8

Summary

Electronic communication is expanding exponentially and systems utilizing cryptographic protocols are the predominant option used to secure the information being passed, and to authenticate principals. New protocols are generated frequently. These cryptographic protocols, designed to ensure the security of encrypted messages or for authentication, may be themselves vulnerable to compromise by persistent intruders, even if the cryptographic algorithm they use is strong. Based on the seminal papers by Needham and Schroeder [NS78] and Burrows, Abadi, and Needham [BAN89] a new research area emerged focused on providing mechanisms to evaluate the ability of cryptographic protocols to meet their stated goals.

8.1 Review of the Research

The research presented in this dissertation represents a new approach to cryptographic protocol verification. Because of the similarity between cryptographic protocols and computer programs, we were lead to research surrounding developing formal methods for program verification. We were particularly interested in the precondition/postcondition reasoning of C. A. R. Hoare [HOAR69] and the weakest precondition predicate transformer of Edsger Dijkstra [DIJK76].

The first task was to develop a formal language that was sufficiently simple to allow a compact formal semantics, yet expressive enough to allow specification of most cryptographic protocols. We developed the language CPAL for this purpose and wrote a supporting syntax checker.

We developed a formal semantics for CPAL using Weakest Preconditions (WP). We began this stage of research by defining the model of computation to represent the fundamental concepts of the system. We identified secure, local address spaces for each prin-

cipal with input queues representing the communications medium for transmission of information between principals. We borrowed concepts from Abadi and Tuttle [AT91] and placed an all powerful intruder at the center of the communications hub, thus requiring the cryptographic system to provide all elements of security.

Once a verification condition generator was in place, it soon became evident that the size of the verification conditions would quickly become a limiting factor in utilizing CPAL as an effective protocol verification tool. Using logical reduction techniques in an automated verification condition simplification routine, we were able to reduce even the longest verification conditions to manageable size.

CPAL-ES, the result of our research, is a complete, fully functional cryptographic protocol evaluation system. CPAL-ES includes a syntax checker for CPAL, a verification condition generator that translates a protocol specified in CPAL into a verification condition expressed in predicate calculus, and a simplification component that reduces common tautologies. CPAL-ES is unique in that it is formal, providing concise definitions for a broad class of protocols based on the syntax of the protocol specification.

8.2 Dissertation Review

In this dissertation, we described the need for communication security founded on cryptography and cryptographic protocols. We gave numerous protocols that illustrate the purpose of cryptographic protocols and that highlight the subtle nature of protocol flaws.

We then reviewed the literature regarding cryptographic protocols, gave and dissected the classic replay flaw detected in the Needham and Schroeder Private Key Protocol [NS78], and reviewed the current research in cryptographic protocol verification. We discussed the differences in the methods of Kemmerer, Meadows, and Millen [KMM93] and reviewed a number of different logics developed to pinpoint weakness in protocols.

As the focus of the dissertation, we describe CPAL-ES as a new tool for analyzing cryptographic protocols based on previous research regarding formal methods of computer program verification. We proposed a new language for cryptographic protocol specification and demonstrated the syntax checker developed for the language. We illustrated the features and expressiveness of the language in a large number of protocols of all types.

At the heart of the research, we described the technique we used for establishing the formal semantics of CPAL. We defined and illustrated the Weakest Precondition definitions for each statement type in the language and provided supporting definitions for the ENCRYPT, DECRYPT, CATENATION, and NEW operators. We showed how the technique works on simple examples and on complex protocols of different types.

Later in the dissertation, we illustrated the applicability of CPAL-ES to a broad class of protocols. We addressed one-way and two-way authentication protocols and the more common, key distribution protocols. We considered protocols using private key cryptography, public key cryptography and others using a combination of the two. In every case, CPAL-ES provided a definition for the protocol meaningful in terms of the protocol assumptions, principal's actions, and protocol goals within the operating environment.

In Chapter 7, we extended the use of CPAL-ES by the illustrating the use of CPAL-ES in combination with an epistemic logic. We showed how we can use the CPAL ASSERT and ASSUME statements to reflect concepts in a wide variety of logics using predicate notation. We then showed how CPAL-ES output could be received as input to a formal theorem prover, thereby providing a complete protocol definition system.

8.3 Conclusion

There are many potential extensions to this research. A highly practical tool, for example, would be a translator that takes the CPAL specification of the protocol and pro-

duces the C (or other common language) code to implement the specified protocol. This would provide the protocol designer with a complete protocol implementation system that would take their protocol from a pseudocode version, evaluate it with regard to its environment and goals, and mechanically produce executable code that exactly implements the design.

It may also be useful to provide an interface from this tool to other existing tools for protocol evaluation. Potentially, this tool could one day be part of a protocol evaluation workbench which includes our specification/verification system, a testing system (such as Interrogator [MCF87]), a BAN implementation (such as [CHEN90]), etc. Another sub-component of such a system would be a language translator to take a protocol specified in CPAL and convert it to the language used by Interrogator to provide a fully integrated environment. For now, CPAL-ES is a fully functional system written in C, and executing in the Sparc 4 and Windows 3.1 environments.

As we described above, CPAL-ES provides the cryptographic protocol analyst a complete protocol verification system. As an analyst's tool, it is fast and easy to use, giving immediate feedback and allowing the analyst to quickly compose "what if" scenarios with protocols to determine the impact of friendly or unfriendly actions on the message-based goals.

Because of the nature of CPAL-ES, we believe it resolves many of the weaknesses of the current ad hoc standard tools for protocol evaluation. CPAL-ES replaces the vague and error-prone idealization process of BAN Logic-type approaches by automatically translating the procedural description of the protocol given in CPAL into a logical definition in a simple predicate logic. The WP definition of statement catenation allows CPAL-ES to force sequencing on the steps in each protocol, resolving the step permutation problem of BAN Logic [SNEK91]. Moreover, CPAL-ES does not attempt to replace BAN Logic, but complements BAN Logic by analyzing protocols using BAN Logic constructs

and rules of inference.

CPAL-ES provides a formal definition for any combination of protocol steps. Because it is formal and automated, this method can be used to model active attacks by systematically attempting attacks against protocols. In this way, CPAL-ES formalizes the notion of intruder actions in formulating a wide variety of attacks against cryptographic protocols.

Finally, CPAL-ES is easily combined with the many epistemic logics proposed in recent literature and a formal theorem prover to form a complete protocol verification system at a greater abstraction level than CPAL-ES alone.

Appendix A

Operational Model for CPAL

Value. A value is a bit stream that may be represented in a computer. There are five value types in this model which may be recursively defined: Named values, encrypted/ decrypted values, catenated values, dot values, and function values

Message. A message is a value that is sent or received across the network.

Principal. A principal is a process. Principals are assumed to know protocol steps. Principals are identified by a bit stream (name) unique to that principal. Resources available to principals are:

- Substantial, but not unlimited, processing power,
- Substantial, but not unlimited, private memory
- Transmission medium for sending messages bit by bit
- Message queue for receiving and temporarily storing messages

Intruder. The intruder is a principal that may receive messages sent by other principals. The intruder may be thought of as an all-powerful intruder if all messages are sent using the insecure send operator. Otherwise, the intruder has no greater power than any other principal and cannot see into any other principal's private address space. The intruder may be used by the analyst to model a wide variety of attacks by copying messages from valid sessions for later replay, starting sessions erroneously, intercepting messages, and modifying messages.

Private memory. Private memory may also be considered to be a private address space. No other principal can detect the contents of this memory. In this large memory space, principals may store, modify, and retrieve values represented as bit streams such as:

- Independently created messages,
- Copies of messages received from another principal,

- Edited messages
- State information related to the protocol
- Identifiers of other principals
- Any other bit stream the principal desires to store

Catenate. The value catenation operator is the infixed comma, e.g. M1,M2. The result of catenating values is to combine two or more values into one value. The original values may be recreated by any principal having the catenated value in their address space.

Encryption. Encryption is performed on a value. The value to be encrypted may be the catenation of two or more values. The result of encrypting a value is to produce another value, with the characteristic that decryption of the encryption using the same key will reproduce the value that was originally encrypted.

Decryption. The result of decrypting a value is to produce another value. The result of decrypting a value which was previously encrypted under the same key is to reproduce the value that was originally encrypted.

Assignment. Simple assignment, with a single destination (left side) identifier, is nondestructive, pass by value assignment. If the source (right side) also contains a single identifier, the source value is considered copied into the memory location identified by the destination identifier. If the source contains multiple identifiers separated by the catenation operator, the values are catenated and the resulting value is stored in the destination memory location. For compound assignment (two or more destination identifiers), the assignment operation must "uncatenate" or separate catenated values. Each catenated source value is copied into the memory location designated by the destination identifiers. Source values are bound to destination names by order, with the first value in the catenation copied into the location designated by the first (leftmost) destination identifier. The number of catenated values must equal the number of destination identifiers or the operation will fail.

Send. Principals may (nondestructively) transmit messages stored in their private memory to other principals. A principal's identifier is synonymous with their address. The result of a normal send operation is that the message will be appended to the message queue of the intended recipient and the corresponding queue counter will be incremented. The result of an insecure send operation is that the message will be appended to the message queue of the intruder and the corresponding queue counter will be incremented. Catenated messages are transmitted, and will arrive in the appropriate queue, as a single message.

Receive. Principals may receive messages which have been appended to their message queue. Receive is a blocking operation. If a message is in the queue when a receive operation is executed, the message is copied to the principal's private memory and deleted from the queue, and the queue counter is decremented. If the message queue counter is zero when a receive is executed, block will occur until a message is appended to the queue.

Transmission medium. Principals send all outgoing messages, bit by bit, via a single output channel. No assumptions are made regarding the privacy of messages on the transmission medium or the message queues.

Message Queue. Principals can only receive messages that are stored on their [FIFO] message queue. The intruder can place messages on other principals message queue at any time. Other principals may update queues using the secure send operations. Valid queue operations are push and pop. The push operation will append a message to the queue and increment the message counter, while the pop operation will copy the next message to be read to the private memory of the principal, delete the entry from the queue and decrement the message counter.

End of Appendix A

Appendix B

CPAL Syntax

```
%token  DIGIT LETTER ID NUMBER IF THEN ELSE ASSERT ACCEPT RJECT
NEW CHAR CAT VC SND SSND REC SREC EQ NE AND OR NOT IMP CO LITER
ASSIGN MINUS ENC DEC COMPASSIGN FUNC LT GT UD COMP STR COND VA
BOOL VALUE DOT SPECIAL GASSERT ASSUME ENC_P DEC_P

%left   '|'
%left   '&'
%left   '+'  '-'
%left   '*'  '/'
%left UMINUS    /* supplies precedence for unary minus */

%%   /* beginning of rules section  */

protocol:       actions ;
actions :       action
          |     actions
                     action;
action  :       ID  ':' stmt ;
stmt    :       error ';'
          |     ASSERT cond ';'
          |     GASSERT gcond ';'
          |     ASSUME gcond ';'
          |     RJECT ';'
          |     SND ID  '(' value ')' ';'
          |     SSND ID '(' value ')' ';'
          |     REC  '(' value ')' ';'
          |     SREC  ID '>' ID  '(' value ')' ';'
          |     IF '(' cond ')' THEN '{' stmts '}'  else_part
          |     ID ASSIGN value ';'
          |     '(  ids ')' ASSIGN  value ';' /*compound assignment*/
          |     '{' stmts  '}' ;
 stmts   :      stmt
          |     stmts  stmt ;
 else_part:
          |     ELSE '{'  stmts  '}' ;
value   :       ID
          |     ENC value ']' ID    /*encryption of messages*/
          |     DEC value ']' ID
          |     ENC_P value ']' ID    /*Public key encryption of messages*/
```

```
        |     DEC_P value ']' ID
        |     ID   '(' values ')'
        |     NEW
        |     '<' values '>'  ;
values  :     value
        |     values ',' value;
ids     :     ID
        |     ids  ',' ID;
cond    :     ID  parameters   /* cond returns a predicate */
        |     LITER
        |     value comp value
        |     neg cond
        |     '(' cond ')'
        |     cond oper cond;
parameters:
        |     '(' parms ')';
parms   :     parm
        |     parms ',' parm;
parm    :     cond ;
gcond   :     gid comp gid
        |     gid gparameters
        |     LITER
        |     neg gcond
        |     '(' gcond ')'
        |     gcond oper gcond ;
gid:          ID '.'  ID;
gparameters:
        |     '(' gparms ')';
gparms  :     gparm
        |     gparms ',' gparm;
gparm   :     gcond;
oper    :     AND
        |     OR
        |     IMP;
neg     :     NOT ;
comp    :     EQ
        |     NE
        |     LT
        |     GT;
```

Appendix C

Cryptographic Protocols Expressed in CPAL

Needham and Schroeder Private Key Protocol [NEED78]

```
S: =>A(S.kas);
A: <-(A.kas);
S: =>B(S.kbs);
B: <-(B.kbs);
A: =>S(e[<A,B,Na>]kas);
S: <-(msg);
S: (A,B,Na) := d[msg]kas;
S: =>A(e[<Na,B,kab,e[<kab,A>]kbs>]kas);
A: <-(msg3);
A: temp2 := d[msg3]kas;
A: (N'a,B,kab,ticket) := temp2;
A: assert(N'a==Na);
A: =>B(ticket);
B: <-(ticket);
B: (kab,A) := d[ticket]kbs;
B: =>A(e[Nb]kab);
A: <-(msg4);
A: Nb := d[msg4]kab;
A: =>B(e[f(Nb)]kab);
B: <-(msg5);
B: N'b := d[msg5]kab;
B: assert(N'b == f(Nb));
```

DENNING & SACCO PRIVATE KEY PROTOCOL [DS81]

```
S: =>A(S.kas);
A: <-(A.kas);
S: =>B(S.kbs);
B: <-(B.kbs);
A: =>S(<A,B>);
S: <-(msg);
S: (A,B) := msg;
S: kab := new;
S: T1 := new;
S: =>A(e[<B,kab,T1,e[<T1,kab,A>]kbs>]kas);
A: <-(Msg1);
A:  tmp := d[Msg1]kas;
A: (dst,kab,T1,Ticket) := tmp;
A: assert(dst == B);
--A: assert(sent(S,kab));
A: =>B(Ticket);
B: <-(Ticket);
B: tmp1 := d[Ticket]kbs;
B: (T1,kab,A) := tmp1;
B: gassert(B.T1 == S.T1);
B: =>A(e[N1]kab);
A: <-(Msg2);
A: N1 := d[Msg2]kab;
A: =>B(e[f(N1)]kab);
B: <-(Msg3);
B: N1' := d[Msg3]kab;
B: assert(N1' == f(N1));
```

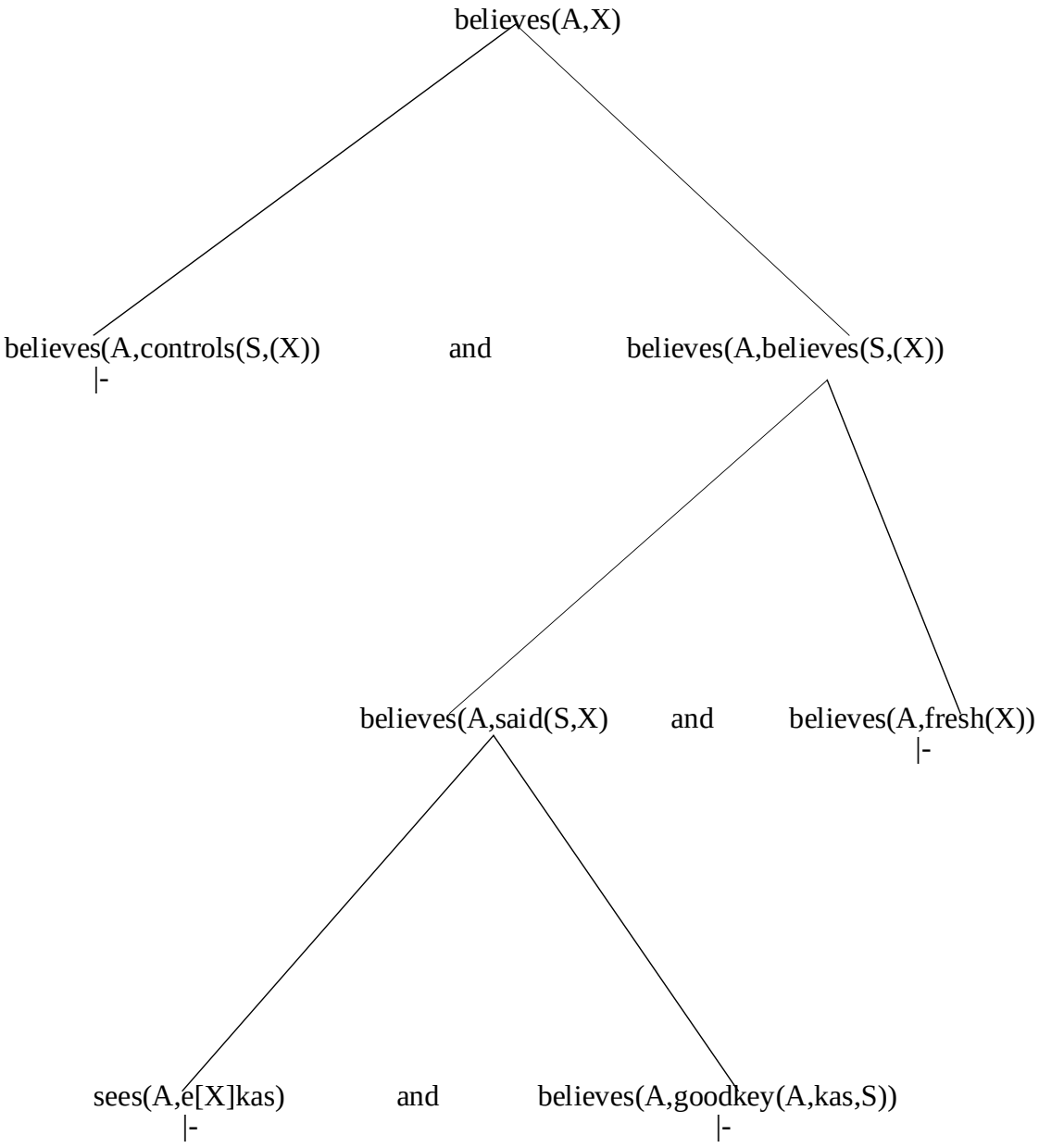
OTWAY AND REES PRIVATE KEY PROTOCOL [OTWY87]

```
S: =>A(S.kas);
A: <-(A.kas);
S: =>B(S.kbs);
B: <-(B.kbs);
A: =>B(<C,A,B,e[<Na,C,A,B>]kas>);
B: <-(msg1);
B: (C,src,dst,Ticket1) := msg1;
B: Ticket2 := e[<Nb,C,A,B>]kbs;
B: =>S(<C,src,dst,Ticket1,Ticket2>);
S: <-(msg2);
S: (C,src,dst,Ticket1,Ticket2) := msg2;
S: (Na,Ca,srca,dstA) := d[Ticket1]kas;
S: assert(C == Ca);
S: (Nb,Cb,srcb,dstB) := d[Ticket2]kbs;
S: assert(C == Cb);
S: passa := e[<Na,kab>]kas;
S: passb := e[<Nb,kab>]kbs;
S: =>B(<C, passa, passb>);
B: <-(msg3);
B: (C',passa, passb) := msg3;
B: assert(C' == C);
B: (Nb',kab) := d[passb]kbs;
B: assert(Nb' == Nb);
B: =>A(<C, passa>);
A: <-(msg4);
A: (C', passa) := msg4;
A: assert(C' == C);
A: (Na',kab) := d[passa]kas;
A: assert(Na' == Na);
```

End of Appendix C

Appendix D

BAN Logic Canonical Proof Tree



Appendix E

Attack on a trivial 2 way protocol [BIRD92]

X: assume(B.k == A.k);

A: ->B(e[<A,e[N1]k>]k);

I: <-(msg1);

I: =>B(msg1);

B: <-(msg1);

B: (src,msg2) := d[msg1]k;

B: N1 := d[msg2]k;

B: N2 := new;

B: msg3 := e[<B,e[N2]k>]k;

B: ->src(<N1,msg3>);

I: <-(msg3a);

I: (N1,msg3) := msg3a;

I: =>A(msg3);

A: <-(msg3);

A: (dst,msg4) := d[msg3]k;

A: N'1 := d[msg4]k;

A: msg5 := e[<A,e[N3]k>]k;

A: ->dst(<N'1,msg5>);

I: <-(msg6);

I: (N2,msg5) := msg6;

I: =>B(N2);

B: <-(N'2);

B: assert(N'2 == N2);

End of Appendix E

Appendix F

PVS Proof of the Needham and Schroeder Private Key Protocol

```
(|gnsban|
 (|gnsgoal| "" (FLATTEN)
  (("" (SPLIT +)
    (("1" (FLATTEN)
      (("1" (SPLIT +)
        (("1"
          (LEMMA "nonce_verification"
           ("P" "B_B" "Q" "B_A" "X" "goodkey(B_A,S_kab,B_B)"))
          (("1" (SPLIT -1)
            (("1" (PROPAX) NIL)
             ("2" (LEMMA "freshness3" ("P" "B_B" "Q" "B_A" "kpq" "S_kab"))
              (("2" (SPLIT -1)
                (("1" (FLATTEN) (("1" (PROPAX) NIL)))
                 ("2" (LEMMA "freshness2" ("P" "B_B" "Y" "B_A" "X" "S_kab"))
                  (("2" (SPLIT -1)
                    (("1" (FLATTEN) (("1" (PROPAX) NIL)))
                     ("2"
                      (LEMMA "freshness1b" ("P" "B_B" "Y" "B_A" "X" "S_kab"))
                      (("2" (SPLIT -1)
                        (("1" (PROPAX) NIL) ("2" (PROPAX) NIL)))))))))))))))
           ("3" (SPLIT -9)
            (("1"
              (LEMMA "msg_meaning1"
               ("P" "B_B" "Q" "B_A" "KPQ" "S_kab" "X" "B_f(B_Nb)"))
              (("1" (SPLIT -1)
                (("1" (PROPAX) NIL)
                 ("2"
                  (LEMMA "jurisdiction"
                   ("P" "B_B" "Q" "B_S" "X" "goodkey(B_B,S_kab,B_A)"))
                  (("2" (SPLIT -1)
                    (("1" (PROPAX) NIL) ("2" (PROPAX) NIL)
                     ("3"
                      (LEMMA "nonce_verification"
                       ("P" "B_B" "Q" "B_S" "X" "goodkey(B_B,S_kab,B_A)"))
                      (("3" (SPLIT -1)
                        (("1" (PROPAX) NIL)
                         ("2"
```

```
                    (LEMMA "freshness3"
                     ("P" "B_B" "Q" "B_A" "kpq" "S_kab"))
                    (("2" (SPLIT -1)
                      (("1" (FLATTEN) (("1" (PROPAX) NIL)))
                       ("2"
                        (LEMMA "freshness2"
                         ("P" "B_B" "Y" "B_A" "X" "S_kab"))
                        (("2" (SPLIT -1)
                          (("1" (FLATTEN) (("1" (PROPAX) NIL)))
                           ("2"
                            (LEMMA "freshness1b"
                             ("P" "B_B" "Y" "B_A" "X" "S_kab"))
                            (("2" (SPLIT -1)
                              (("1" (PROPAX) NIL)
                               ("2" (PROPAX) NIL)))))))))))
                   ("3" (SPLIT -10)
                    (("1"
                      (LEMMA "msg_meaning1"
                       ("P" "B_B" "Q" "B_S" "KPQ" "S_kbs" "X"
                        "S_kab"))
                      (("1" (SPLIT -1)
                        (("1" (PROPAX) NIL) ("2" (PROPAX) NIL)
                         ("3"
                          (LEMMA "sees_enc"
                           ("P" "B_B" "Y" "B_A" "X" "S_kab" "K"
                            "S_kbs"))
                          (("3" (SPLIT -1)
                            (("1" (FLATTEN) (("1" (PROPAX) NIL)))
                             ("2" (PROPAX) NIL)))))))))
                     ("2"
                      (LEMMA "said_sym"
                       ("P" "B_B" "Q" "B_S" "T" "B_B" "R" "B_A" "k"
                        "S_kab"))
                      (("2" (SPLIT -1)
                        (("1" (PROPAX) NIL)
                         ("2" (PROPAX) NIL)))))))))))
              ("3" (PROPAX) NIL)))))
          ("2" (PROPAX) NIL)))))))
     ("2" (POSTPONE) NIL)))))
  ("2"
```

```
(LEMMA "nonce_verification"
 ("P" "A_A" "Q" "A_B" "X" "goodkey(A_A,S_kab,A_B)"))
(("2" (SPLIT -1)
  (("1" (PROPAX) NIL)
   ("2" (LEMMA "freshness3" ("P" "A_A" "Q" "A_B" "kpq" "S_kab"))
    (("2" (SPLIT -1)
      (("1" (FLATTEN) (("1" (PROPAX) NIL)))
       ("2" (LEMMA "freshness2" ("P" "A_A" "X" "A_Na" "Y" "S_kab"))
        (("2" (SPLIT -1)
          (("1" (FLATTEN) (("1" (PROPAX) NIL)))
           ("2" (LEMMA "freshness1" ("P" "A_A" "X" "A_Na" "Y" "S_kab"))
            (("2" (SPLIT -1)
              (("1" (PROPAX) NIL) ("2" (PROPAX) NIL)))))))))))
   ("3" (SPLIT -9)
    (("1"
      (LEMMA "msg_meaning1"
       ("P" "A_A" "Q" "A_B" "KPQ" "S_kab" "X" "A_Nb"))
      (("1" (SPLIT -1)
        (("1" (PROPAX) NIL)
         ("2"
          (LEMMA "jurisdiction"
           ("P" "A_A" "Q" "A_S" "X" "goodkey(A_A,S_kab,A_B)"))
          (("2" (SPLIT -1)
            (("1" (PROPAX) NIL) ("2" (PROPAX) NIL)
             ("3"
              (LEMMA "nonce_verification"
               ("P" "A_A" "Q" "A_S" "X" "goodkey(A_A,S_kab,A_B)"))
              (("3" (SPLIT -1)
                (("1" (PROPAX) NIL)
                 ("2"
                  (LEMMA "freshness3"
                   ("P" "A_A" "Q" "A_B" "kpq" "S_kab"))
                  (("2" (SPLIT -1)
                    (("1" (FLATTEN) (("1" (PROPAX) NIL)))
                     ("2"
                      (LEMMA "freshness2"
                       ("P" "A_A" "X" "A_Na" "Y" "S_kab"))
                      (("2" (SPLIT -1)
                        (("1" (FLATTEN) (("1" (PROPAX) NIL)))
                         ("2"
```

```
                (LEMMA "freshness1"
                 ("P" "A_A" "X" "A_Na" "Y" "S_kab"))
                (("2" (SPLIT -1)
                  (("1" (PROPAX) NIL)
                   ("2" (PROPAX) NIL)))))))))))))
          ("3" (PROPAX) NIL)))))))))
    ("3" (POSTPONE) NIL)))))
("2" (POSTPONE) NIL))))))))))))
```

End of Appendix F

Appendix G

PVS Proof of Snekkenes KP Protocol

```
(|gkpban2|
 (|gkpgoal| "" (FLATTEN)
  (("" (SPLIT 1)
    (("1"
      (LEMMA "nonce_verification"
       ("P" "B_B" "Q" "B_A" "X" "goodkey(B_A,S_kab,B_B)"))
      (("1" (SPLIT -1)
        (("1" (PROPAX) NIL)
         ("2" (LEMMA "freshness3" ("P" "B_B" "Q" "B_A" "kpq" "S_kab"))
          (("2" (SPLIT -1)
            (("1" (FLATTEN) (("1" (PROPAX) NIL)))
             ("2" (LEMMA "freshness2" ("P" "B_B" "Y" "B_Nb" "X" "S_kab"))
              (("2" (SPLIT -1)
                (("1" (FLATTEN) (("1" (PROPAX) NIL)))
                 ("2" (LEMMA "freshness1b" ("P" "B_B" "Y" "B_Nb" "X" "S_kab"))
                  (("2" (SPLIT -1)
                    (("1" (PROPAX) NIL) ("2" (PROPAX) NIL)))))))))))))
        ("3" (SPLIT -19)
         (("1"
           (LEMMA "msg_meaning1"
            ("P" "B_B" "Q" "B_S" "KPQ" "S_kbs" "X" "S_kab"))
           (("1" (SPLIT -1)
             (("1" (PROPAX) NIL) ("2" (PROPAX) NIL)
              ("3" (LEMMA "sees_enc"
                ("P" "B_B" "X" "cat(S_kab,A_A)" "Y"
                 "cat(B_Nb,e(cat(A_Na,S_kab),S_kas))" "K" "S_kbs"))
               (("3" (SPLIT -1)
                 (("1" (FLATTEN)
                   (("1" (LEMMA "sees_enc"
                     ("P" "B_B" "X" "S_kab" "Y" "A_A" "K" "S_kbs"))
```

```
            (("1" (SPLIT -1)
              (("1" (FLATTEN) (("1" (PROPAX) NIL)))
               ("2" (PROPAX) NIL)))))))
          ("2" (PROPAX) NIL)))))))))
 ("2" (SPLIT -18)
  (("1"
    (LEMMA "msg_meaning1"
     ("P" "B_B" "Q" "B_A" "KPQ" "S_kab" "X" "B_Nc"))
    (("1" (SPLIT -1)
      (("1" (PROPAX) NIL)
       ("2"
        (LEMMA "jurisdiction"
         ("P" "B_B" "Q" "B_S" "X" "goodkey(B_B,S_kab,B_A)"))
        (("2" (SPLIT -1)
          (("1" (PROPAX) NIL) ("2" (PROPAX) NIL)
           ("3"
            (LEMMA "nonce_verification"
             ("P" "B_B" "Q" "B_S" "X" "goodkey(B_B,S_kab,B_A)"))
            (("3" (SPLIT -1)
              (("1" (PROPAX) NIL)
               ("2"
                (LEMMA "freshness3"
                 ("P" "B_B" "Q" "B_A" "kpq" "S_kab"))
                (("2" (SPLIT -1)
                  (("1" (FLATTEN) (("1" (PROPAX) NIL)))
                   ("2"
                    (LEMMA "freshness2"
                     ("P" "B_B" "Y" "B_Nb" "X" "S_kab"))
                    (("2" (SPLIT -1)
                      (("1" (FLATTEN) (("1" (PROPAX) NIL)))
                       ("2"
                        (LEMMA "freshness1b"
                         ("P" "B_B" "Y" "B_Nb" "X" "S_kab"))
                        (("2" (SPLIT -1)
```

```
                    (("1" (PROPAX) NIL)
                     ("2" (PROPAX) NIL)))))))))))))
                ("3"
                 (LEMMA "said_sym"
                  ("P" "B_B" "Q" "B_S" "T" "B_B" "R" "B_A" "k"
                   "S_kab"))
                 (("3" (SPLIT -1)
                   (("1" (PROPAX) NIL) ("2" (PROPAX) NIL)))))))))))))
            ("3"
             (LEMMA "sees_enc"
              ("P" "B_B" "Y" "A_A" "X" "B_Nc" "K" "S_kab"))
             (("3" (SPLIT -1)
               (("1" (FLATTEN) (("1" (PROPAX) NIL)))
                ("2" (PROPAX) NIL)))))))))
        ("2" (PROPAX) NIL)))))))))
("2"
 (LEMMA "nonce_verification"
  ("P" "A_A" "Q" "A_B" "X" "goodkey(A_A,S_kab,A_B)"))
 (("2" (SPLIT -1)
   (("1" (PROPAX) NIL)
    ("2" (LEMMA "freshness3" ("P" "A_A" "Q" "A_B" "kpq" "S_kab"))
     (("2" (SPLIT -1)
       (("1" (FLATTEN) (("1" (PROPAX) NIL)))
        ("2" (LEMMA "freshness2" ("P" "A_A" "X" "A_Na" "Y" "S_kab"))
         (("2" (SPLIT -1)
           (("1" (FLATTEN) (("1" (PROPAX) NIL)))
            ("2" (LEMMA "freshness1" ("P" "A_A" "X" "A_Na" "Y" "S_kab"))
             (("2" (SPLIT -1)
               (("1" (PROPAX) NIL) ("2" (PROPAX) NIL)))))))))))))
    ("3" (SPLIT -18)
     (("1"
       (LEMMA "msg_meaning1"
        ("P" "A_A" "Q" "A_B" "KPQ" "S_kab" "X" "A_Na"))
       (("1" (SPLIT -1)
```

```
((("1" (PROPAX) NIL)
 ("2"
  (LEMMA "jurisdiction"
   ("P" "A_A" "Q" "A_S" "X" "goodkey(A_A,S_kab,A_B)"))
  (("2" (SPLIT -1)
    (("1" (PROPAX) NIL) ("2" (PROPAX) NIL)
     ("3"
      (LEMMA "nonce_verification"
       ("P" "A_A" "Q" "A_S" "X" "goodkey(A_A,S_kab,A_B)"))
      (("3" (SPLIT -1)
        (("1" (PROPAX) NIL)
         ("2"
          (LEMMA "freshness3"
           ("P" "A_A" "Q" "A_B" "kpq" "S_kab"))
          (("2" (SPLIT -1)
            (("1" (FLATTEN) (("1" (PROPAX) NIL)))
             ("2"
              (LEMMA "freshness2"
               ("P" "A_A" "X" "A_Na" "Y" "S_kab"))
              (("2" (SPLIT -1)
                (("1" (FLATTEN) (("1" (PROPAX) NIL)))
                 ("2"
                  (LEMMA "freshness1"
                   ("P" "A_A" "X" "A_Na" "Y" "S_kab"))
                  (("2" (SPLIT -1)
                    (("1" (PROPAX) NIL)
                     ("2" (PROPAX) NIL)))))))))))))
       ("3" (SPLIT -19)
        (("1"
          (LEMMA "msg_meaning1"
           ("P" "A_A" "Q" "A_S" "KPQ" "S_kas" "X" "S_kab"))
          (("1" (SPLIT -1)
            (("1" (PROPAX) NIL) ("2" (PROPAX) NIL)
             ("3"
```

```
                    (LEMMA "sees1"
                     ("P" "A_A" "X" "e(cat(A_Na,S_kab),S_kas)" "Y"
                      "e(cat(A_Na,B_Nc),S_kab)"))
                    (("3" (SPLIT -1)
                      (("1" (FLATTEN)
                        (("1"
                          (LEMMA "sees_enc"
                           ("P" "A_A" "X" "A_Na" "Y" "S_kab" "K"
                            "S_kas"))
                          (("1" (SPLIT -1)
                            (("1" (FLATTEN) (("1" (PROPAX) NIL)))
                             ("2" (PROPAX) NIL)))))))
                       ("2" (PROPAX) NIL)))))))))
                ("2" (PROPAX) NIL))))))))))
           ("3" (LEMMA "sees1"
                 ("P" "A_A" "X" "e(cat(A_Na,S_kab),S_kas)" "Y"
                  "e(cat(A_Na,B_Nc),S_kab)"))
                (("3" (SPLIT -1)
                  (("1" (FLATTEN)
                    (("1"
                      (LEMMA "sees1"
                       ("P" "A_A" "X" "e(cat(A_Na,S_kab),S_kas)" "Y"
                        "e(cat(A_Na,B_Nc),S_kab)"))
                      (("1" (SPLIT -1)
                        (("1" (FLATTEN)
                          (("1"
                            (LEMMA "sees_enc"
                             ("P" "A_A" "X" "A_Na" "Y" "B_Nc" "K" "S_kab"))
                            (("1" (SPLIT -1)
                              (("1" (FLATTEN) (("1" (PROPAX) NIL)))
                               ("2" (PROPAX) NIL)))))))
                         ("2" (PROPAX) NIL)))))))
                   ("2" (PROPAX) NIL)))))))))
          ("2" (PROPAX) NIL))))))))))))
```

Bibliography

[AN94] Martin Abadi and Roger Needham, "Prudent Engineering Practice for Cryptographic Protocols", From the 1994 IEEE Computer Society Symposium on Research in Security and Privacy, pp. 122-136

[AT91] Martin Abadi and Mark R. Tuttle, "A Semantics for a Logic of Authentication", Tenth Annual ACM Symp on Princ of Dist Computing, Montreal, Canada, August, 1991

[BAN88] Burrows, M., Abadi, M., and Needham, R. M. "A Practical Study in Belief and Action", In Proceedings of the 2nd Conference on Theoretical Aspects of Reasoning about Knowledge (Asilomar, Ca., Feb. 1988) M. Vardi, Ed. Morgan Kaufmann, Los Altos, Calif., 1988, pp. 325-342

[BAN89] Burrows, M., Abadi, M., and Needham, R. M. "A Logic of Authentication", Technical Report 39, Digital Systems Research Center, 130 Lytton Avenue, Palo Alto, Cal, 94301, February 1989

[BAN90] Burrows, M., Abadi, M., and Needham, R. M. "A Logic of Authentication", ACM Transactions on Computer Systems, Vol. 8, No. 1, Feb 1990, pp. 18-36.

[BAN90b] Burrows, M., Abadi, M., and Needham, R. M., "Rejoinder to Nessett", ACM Operating Systems Review, vol. 24, no. 2, April 1990, pp. 39-40

[BELMER90] Steven M. Bellovin & Michael Merritt, "Limitations of the Kerberos Authentication System", Computer Communications Review, Oct 1990

[BIEB90] P. Bieber, "A Logic of Communication in a Hostile Environment", in Proceedings of the ComputereSecurity Foundations Workshop III, Washington (IEEE), 1990, pp.14-22

[BIRD92] Ray Bird, Inder Gopal, Amir Herzberg, Phil Janson, Shay Kutten, Refik Molva, and Moti Yung. "Systematic Design of Two-Party Authentication Protocols." In Joan Fegenbaum, editor, Advances in Cryptography - CRYPTO `91, volume 576 of Lecture Notes in Comuter Science. Springer Verlag, Berlin, 1992

[BIRD93] Ray Bird, Inder Gopal, Amir Herzberg, Phil Janson, Shay Kutten, Refik Molva, and Moti Yung. "Systematic Design of a Family of Attack Resistant Authentication Protocols", IEEE Journal on Selected Areas in Communications, Vol 11, No. 5, June 1993

[BM79] Robert S. Boyer and J. Strother Moore, "A Computational Logic", Academic Press 1979, From the ACM Monographic Series

[BM88] Robert S. Boyer and J. Strother Moore, "A Computational Logic Handbook", Academic Press 1988, From the series "Perspectives in Computing", v23

[CARL94] Ulf Carlsen, "Generating Formal Cryptographic Protocol Specifications", From the 1994 IEEE Computer Society Symposium on Research in Security and Privacy, pp 137-145

[CCITT] CCITT draft recommendation X.509. The directory-authentication framework, version 7. CCITT, Gloucester, Nov. 1987

[CHEN90] Cheng, Pau-Chen and Gligor, Virgil D. "On the formal specification and verification of a Multiparty Session Protocol". From 1990 IEEE Computer Society Symposium on Research in Security and Privacy, pp. 216-233

[CM81] Clockson, W. F., and Mellish, C. S., "Programming in Prolog", Springer-Verlag 1981

[CORSS] Judy Crow, Sam Owre, John Rushby, Natarajan Shankar, Mandayam Srivas, "A Tutorial Introduction to PVS", Computer Science Laboratory, SRI International, Menlo Park, Ca 94025

[DES77] National Bureau of Standards (NBS). Data Encryption Standard. Dederal Information Processing Standard, Publication 46, NBS, Washington, D.C., January 1977

[DH76] W. DIffie, M. Hellman, New Directions in Cryptography, IEEE Transactions on Information Theory, Vol. it-22, no 6, Nov 1976, pp. 644-654

[DIFF79] Diffie, W., Hellman, M.E. 'Privacy and Authentication: an Introduction to Cryptography' Proceedings of the IEEE, v67 n3, March 1979, pp 397-427.

[DIFF88] Diffie, W., "The First Ten Years of Public Key Cryptography", Proceedings of The IEEE, Vol 76, No. 5, May 1988, pp 560-577

[DIJK76] Edsger W. Dijkstra, "A Discipline of Programming", Prentice Hall Series in Automatic Computation, Prentice-Hall Inc. Englewood Cliffs, NJ, 1976

[DOL83] Dolev, D., and Yao, A.C. "On the security of public key protocols". IEEE Trans. Inf. Theory IT-29, 2(Mar. 1983), pp. 198-208

[DS81] D. E. Denning and G. M. Sacco, "Timestamps in key distribution protocols," Communications of the ACM, vol. 24, no. 8, Aug 1981, pp. 533-536

[ELGAM88] T. ElGamal, "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms", IEEE Transactions on Information Theory, IT-31 (4):469-472

[GLAS88] Janice I. Glasgow and Glenn H. MacEwen, "Reasoning About Knowledge in Multilevel secure Distributed Systems", in Proceedings of the 1988 IEEE Sypposium on Security and Privacy, Washington (IEEE), 1988, pp. 122-128

[GMW86] O. Goldreich, S. Micali, and A. Wigderson, "Proofs That Yield Nothing but Their Validity and a Methodology of Cryptographic Protocol Design", Proceedings

of the 27th IEEE Symposium on foundations of Computer Science, 1986, pp. 174-187

[GNY90] Gong, L., Needham, R., and Yahalom, R. "Reasoning about Belief in Cryptographic protocols". From 1990 IEEE Computer Society Symposium on Research in Security and Privacy, pp. 234-248

[GOLD85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff, "The Knowledge Complexity of Interactive Proof Systems," Proc. 27th Annual IEEE Symposium on Foundations of Computer Science, 1985, pp. 291-304

[GOLD89] Goldwasser, Shafi, Micali, Silvio, and Rackoff, Charles, "The Knowledge Complexity of Interactive Proof Systems," Siam Jrnl of Comp, Vol 18, No 1, Feb 1989, pp. 186-208.

[GONG93] Li Gong, "Increasing Availability and Security of an Authentication Service", IEEE Journal on Selected Areas in Communications, Vol 5, No. 11, 5 June 1993, pp657-662

[GOOD78] D I. Good et. al. Report on the language Gypsy - version 2.0, Univ. of Texas at Austin, Certifiable Minicomputer Project, Report ICSCA-CMP-10, 1978

[GORD84] J. Gordon, speech at the Zurich Seminar, 1984. In this lecture, which has unfortunately never been published [but was reported by Whitfield Diffie], Gordon assembled the facts of Alice and Bob's precarious lives, which had previously been available only as scattered references in the literature.

[GS91] Klaus Gaarder and Einar Snekkenes, "Applying a Formal Analysis Technique to the CCITT X.509 Strong Two-Way Authentication Protocol", Journal Of Cryptology, 3:81-98, 1991.

[GUMB82] Raymond D. Gumb, "On the Underlying Logics of Specification Languages", ACM Sigsoft, Software Engineering Notes, Vol 7, No 4, Oct 82, pp 21-23

[HAL88] Halpern, J.Y., and Moses, Y.O. "A Knowledge-based analysis of zero knowledge" (preliminary report). In Proceedings of the 20th ACM symposium on Theory of Computing (CHicago, Ill, May 1988), ACM, New York, 1988, pp. 132-147

[HOAR69] C.A.R. Hoare, "An Axiomatic Basis for Computer Programming", Communications of the ACM, Vol 12, Number 10, Oct 1969

[HOAR73] Hoare, C. A. R., and N. Wirth: "An Axiomatic Definition of the Programming Language Pascal," Acta Informatica, 2, 1973

[HOAR78] C. A. R. Hoare, "Communicating Sequential Processes", Communications of the ACM, Vol 21, Number 8, Aug 1978, pp 666-677

[HOAR85] C. A. R. Hoare, "Communicating Sequential Processes", Prentice Hall, 1985

[KAHN67] D. Kahn, "The Codebreakers, The Story of Secret Writing", New York: Mac-

Millan, 1967

[KEM89] R. A. Kemmerer, "Using Formal Methods to Analyze Encryption Protocols," IEEE Journal on Selected Areas in Communications, vol. 7, mo. 4, pp. 448-457, May 1989

[KG91] Rajashekar Kailar and Virgil D. Gligor. "On Belief Evolution n Authentication Protocols", In Proceedings og the Computer Security Foundations Workshop IV, pp. 103-116, IEEE Computer Society Press, Los Alamitos, Ca. 1991

[KMM93] R. Kemmerer, C. Meadows, and J. Millen, "Three Systems for Cryptographic Protocol Analysis", To appear in The Journal of Cryptography

[LAMP91] B. Lampson, M. Abadi, M. Burrows, and E. Wobber, "Authentication in Distributed Systems: Theory and Practice", ASM OS Review, Vol 25, No. 5, Special Issue, Proceedings of the 13th Symposium on Operating System Principles, 13-16 Oct 1991, pp 165-182

[LGS89] Mark Lomas, Li Gong, Jerome H. Saltzer, Roger Needham, "Reducing Risks from Poorly Chosen Keys", Operating Systems Review, 12th ACM Symposium on Operating Sstems Principles, Vol 23, Number 5, 3-6 Dec 1989, p 14-18

[MB93] Wenbo Mao and Colin Boyd, "Towards a Formal Analysis of Security Protocols", In Proceedings of the Computer Security Foundations Workshop VI, pp 147-158, IEEE Computer Society Press, Los Alamitos, California, 1993

[MCF87] Millen, J.K., Clark, S. C., and Freedman, S. B. "The interrogator: Protocol security analysis". IEEE Trans. Sofw. eng. SE-13, 2(Feb. 1987), pp. 274-288

[MEAD89] Meadows, C., "Using Narrowing in the Analysis of Key Management Protocols". From 1989 IEEE Computer Society Symposium on Research in Security and Privacy, pp. 138-147.

[MEAD91] Meadows, C., "A System for the Specification and Analysis of Key Management Protocols". From 1991 IEEE Computer Society Symposium on Research in Security and Privacy, pp. 182-195.

[MEAD92] Meadows, C., "Applying Formal Methods to the Analysis of Key Management Protocols", Journal of Computer Security, Vol. 1, No. 1, 1992

[MER78] Merkle, R. C., 'Secure Communications over Insecure Channels', Communications of the ACM, v21 n4, April 1978, pp 294-300.

[MNSS87] Miller,S.P., Neuman, C., Schiller, J.I., and Saltzer, J.H., Kerberos Authentication and Authorization System. In Project Athena Technical Plan, MIT, Cambridge, Mass., July 1987

[MOORE88] Judy H. Moore, "Protocol Failures in Cryptosystems", Proceedings of the IEEE, Vol. 76, No. 5, May 1988

[MOS89] L. Moser, "A Logic of Knowledge and Belief for Reasoning about Computer Security" in Proceedings of the Computer Security Foundations Workshop II, Washington (IEEE), 1989, pp. 57-63

[NEED78] Needham, R. M., and Schroeder, M. D. "Using encryption for authentication in large networks of computers". Commun. ACM 21, 12 (Dec. 1978), pp. 993-999

[NEED87] Needham, R.M. & Schroeder, M.D., "Authentication Revisited", ACM Operating Systems Review, Vol. 21, No. 1, January 1987.

[NESS87] D. M. Nessett, "Factors Affecting Distributed System Security", IEEE Trans On SWE, Vol. SE-13, No 2, Feb 1987, pp. 204-222

[NESS89] D. M. Nessett, "Layering Central Authentication on Existing Distributed System Terminal Services", From 1989 IEEE Computer Society Symposium on Security and Privacy, pp. 290-299.

[NESS90] D. Nessett, "A Critique of the Burrows, Abadi, and Needham Logic", ACM Operating Systems Review, vol. 24, no. 2, April 1990, pp. 35-38

[NS93] B. Clifford Neuman and StuartG. Stubblebine. A Note on the Use of Timestamps as Nonces. Operating Systems Review, 27(2):10-14, April 1993

[OSR93a] S. Owre, N. Shankar, and J. M. Rushby. "The PVS Specification Language (Draft). Computer Science Laboratory, SRI International, Menlo Park, CA, February 1993

[OSR93b] S. Owre, N. Shankar, and J. M. Rushby. "User Guide for the PVS Specification and Verification System (Draft). Computer Science Laboratory, SRI International, Menlo Park, CA, March 1, 1993

[OTWY87] Otwy, D., and Rees, O. "Efficient and timely mutual authentication". Operating Systems Review 21, 1(Jan. 1987), pp. 8-10

[RANG88] P. Venkat Rangan, "An Axiomatic Basis for Trust in Distributed Systems", in Proceedings of the 1988 IEEE Symposium on Security and Privacy, pp. 204-211, IEEE Computer Society Press, Washington, DC, 1988

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman, " A Method for Obtaining Digital Signatttures and Public-key Cryptosystems", Communications of the ACM, 21, 2, Feb 1978, 120-126

[SIMM85] Simmons, G.J., "How to (Selectively) Broadcast a Secret," in Proceedings of the 1985 IEEE Symposium on Security and Privacy, IEEE Computer Society Press, 1985, pp. 108-113

[SNEK91] Snekkenes, E., "Exploring the BAN Approach to Protocol Analysis". From 1991 IEEE Computer Society Symposium on Research in Security and Privacy, pp. 171-181.

[SNEK92] Einar Snekkenes, "Roles in Cryptographic Protocols", Proceedings of the IEEE Symposium on Research in Security and Privacy, Oakland, 1992, pp 105-118

[SOR93a] S. Owre, N. Shankar, and J. M. Rushby. "PVS Tutorial. Computer Science Laboratory, SRI International, Menlo Park, CA, February 1993

[SOR93b] S. Owre, N. Shankar, and J. M. Rushby. "The PVS Proof Checker: A Reference Manual". Computer Science Laboratory, SRI International, Menlo Park, CA, March 1, 1993

[SYV90] P. Syverson, "A Logic for Cryptographic Protocol Analysis", NRL Formal Report 9305, December 1990

[SYV91] Syverson, P., "The Use of Logic in the Analysis of Cryptographic Protocols", From 1991 IEEE Computer Society Symposium on Research in Security and Privacy, 156-170

[SYV92] Syverson, P., "Knowledge, Belief, and Semantics in the Analysis of Cryptographic Protocols", Journal of Computer Security 1 (1992), pp 317-334

[SYV93a] Syverson, P., "Adding Time to a Logic of Authentication", in Proceedings of the First ACM Conference on Computer and Communications Security (Fairfax VA, Nov 3-5).

[SYV93b] Syverson, P., "On Key Distribution Protocols for Repeated Authentication" ACM Operating Systems Review vol 27, no 4, October 1993, pp. 24-30.

[SYVOOR94] Syverson, P., and van Oorschot, P.C., "On Unifying Some Cryptographic Protocol Logics", in Proceedings of the 1994 IEEE Computer Societ Symposium on Research in Security and Privacy, May 16-18, 1994 Oakland, California

[TMN91] Tatebayashi, M., N. Mattsuzaki, D. B. Newman, "Key Distribution Protocol for Digital Mobile Communication Systems," in Advances in Cryptology - CRYPTO ’89, LNCS 435, G. Brassard, ed. Springer-Verlag, 1991, pp. 324-333

[vO93] Paul 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, pp 232-243, Nov 1993

[VOY83] Voydoc, V.L., Kent, S.T., 'Security Mechanisms in High-Level Protocols', Computing Surveys, v15 n2, June 1983.

[WL92] T.Y.C. Woo and S.S. Lam. Authentication for Distributed Systems. Computer Vol. 25, No. 1, January 1992, pp. 39-52

[WULF81] Wm. A. Wulf, Mary Shaw, Paul N. Hilfinger, and Lawrence Flon, "Fundamental Structures of Computer Science" Addison-Wesley, 1981

[WULF93] Wm. A. Wulf, Alec Yasinsac, Katie S. Oliver, and Ramesh Peri, "Remote Authentication Without Prior Shared Knowledge", Proceedings of the Internet

Society Symposium on Network and Distributed System Security, Feb 2-4, 1994, San Diego, Ca., pp. 159-164

[YW93] Alec Yasinsac and Wm. A. Wulf, "A Formal Semantics for Evaluating Cryptographic Protocols", University of Virginia, Technical Report # CS-93-53, August 1993