A Systematic Method to Analyze Cryptographic Protocols using Discrete-Event Systems

by

Shaowu Luo

A thesis submitted to the Department of Electrical and Computer Engineering
in conformity with the requirements for
the degree of
Master of Science (Engineering)

Queen's University
Kingston, Ontario, Canada

September 1999

Abstract

As business applications over the Internet proliferate, cryptographic protocols have become a hot issue. Owing to their subtlety, however, the design of cryptographic protocols has proven to be surprisingly error prone. Therefore it is desirable to use formal methods to analyze cryptographic protocols. In this thesis, a systematic method for the analysis of cryptographic protocols is proposed based on the supervisory control framework of Discrete-Event Systems (DESs). The supervisory control framework of DESs is advantageous in handling the analysis of attacks on cryptographic protocols due to its exploitation of the theory of automata and formal languages. The basic operations meet and parallel compositions make the Divide-and-Conquer strategy applicable. An automatic scheme is proposed, producing related models efficiently. Case studies on the Meyer-Matyas key distribution protocol, the Needham-Schroeder authentication protocol, the Tatebayashi-Matsuzaki-Newman key distribution protocol and Netscape's original SSL (Secure Socket Layer) protocol have shown the soundness of the proposed method. In particular, the construction of an attack on Netscape's original SSL protocol appears to be novel. So does the construction of an attack on the Tatebayashi-Matsuzaki-Newman key distribution protocol, and the attack cannot be prevented by the countermeasure discussed by M. Tatebayashi, N. Matsuzaki and D. B. Newman.

To my parents, wife, and daughter

For their love, support, and understanding

Acknowledgments

I would like to thank my supervisor Dr. Karen Rudie for her help, guidance, and encouragement.

Special thanks go to Dr. Henk Meijer and Dr. Stafford Tavares for their guidance and help.

I would like also to thank my classmates and friends for their support and encouragement.

I gratefully acknowledge the financial support of the School of Graduate Studies and Research at Queen’s University and the Department of Electrical and Computer Engineering at Queen’s University.

Contents

List of Figures

List of Tables

1 Introduction

In recent years, business applications over computer networks, in particular the Internet, have been proliferating. As a result, network security has become a hot issue, especially after some security breaks were publicly reported. Cryptography and cryptographic protocols have been recognized as the main means for securing communications over networks. A cryptographic protocol is a sequence of messages between two or more parties in which encryption is used to provide authentication or to distribute cryptographic keys for new conversations. Cryptographic protocols are relatively simple in structure, but their security properties are not always intuitively obvious. A number of protocols that have appeared in the literature and subsequently proved to be flawed suggests that more formal analysis is necessary. In this chapter, we will first present the potential value of this thesis. Then we will describe the context and outline of the project.

1.1 Motivation

As technology advances, computer networks result in more and more applications. They are used in banking for everyday actions such as e-Commerce (Electronic Commerce), Electronic Funds Transfer (EFT), and Automated Teller Machine (ATM) transactions. They are used to store and transfer trade secrets for corporations, and military and diplomatic secrets for governments. According to an International Data Corporation report [1], *Internet Commerce Market Model*, the value of business conducted over the

Internet is expected to grow from $2.6 billion in 1996 to some $220 billion in 2001. Even this almost hundred-fold increase is regarded by some as a conservative prediction. Within just the last three years, a whole new industry of e-Commerce service providers has appeared. Meanwhile, the standards required to run a stable and secure e-Commerce are proliferating and maturing rapidly. However, some serious computer network security issues have been found, such as undetected theft of credit-card data and security lapses in the Internet world-wide-web. Visa International Corporation reported that although transactions over the Internet amounted to only 2% of the total transactions, about half of all transaction disputes resulted from so-called e-Commerce. In fact, related security breaches can take a variety of forms. These include the following:

- An unauthorized person, such as a contractor or visitor, might gain access to a company's computer system.
- An employee or supplier authorized to use the system for one purpose might use it for another. For example, an engineer might break into the human resources database to obtain confidential salary information.
- Confidential information might be intercepted as it is being sent to an authorized user. For example, an intruder might attach a network sniffing device to the network. While sniffers are normally used for network diagnostics, they can also be used to intercept data travelling over the wire.
- Users may share documents between geographically separated offices over the Internet or Extranet, or telecommuters accessing the corporate Intranet from their home computers can expose sensitive data as it is sent over the wire.

- Electronic mail can be intercepted in transit.

These are not merely theoretical concerns. A number of computer hackers breaking into corporate computer systems over the Internet have received a great deal of press in recent years. Passwords are also largely ineffective against inside attacks. Most passwords are notoriously easy to guess. Even when passwords are not guessed, or when more sophisticated access control methods are used, it is important to note that access control alone can not ensure that information remains confidential. The growth in network complexity has increased the potential points of attack both from outside and from within organizations. The TCP/IP (Internet) protocols and technology are inherently designed to be open. The TCP/IP is a connectionless protocol: data is broken up into packets that travel freely over the network, seeking the best possible route to reach their final destination. Therefore, unless proper precautions are taken, data can readily be intercepted and/or altered—often without either the sending or the receiving party being aware of the security breach. Because dedicated links between the parties in a communication are usually not established in advance, it is easy for one party to impersonate another party.

These problems give rise to network security concerns. In order to tackle them, people have been proposing and analyzing a number of technologies in recent years, among which various cryptographic protocols are the major ones. Cryptographic protocols can help establish secure communication channels in an open environment. A cryptographic protocol is a sequence of message exchanges between two or more parties in which encryption is used. To communicate securely over insecure channels, it is essential that the related cryptographic protocols be designed properly, assuming that the basic cryptographic algorithms are intractable. However, the design of cryptographic protocols is a notoriously difficult task and is surprisingly error prone. In fact, it has been shown that it is possible to inadvertently design protocols with subtle security flaws that may go undiscovered for a long time, even if careful effort is committed. There are a number of examples: the Needham-Schroeder key distribution protocol [2] was shown to be vulnerable to several kinds of replay attacks [3,4], and also to a more complicated impersonating attack [5]. A public-key protocol contained in a draft CCITT X.509 standard [6] was shown to have two separate security flaws [7]. The selective broadcast protocol of G. J. Simmons [8] was shown to have a subtle flaw in its authentication mechanism [9]. Some other examples are discussed in [10-15]. Therefore, a systematic method for analysis of cryptographic protocols has significant practical and theoretical value.

1.2 Thesis Outline

Based on the supervisory control framework of discrete-event systems (DESs), this thesis proposes a systematic method to analyze attacks by intruders on cryptographic protocols. An intruder is an evil entity that takes its seat at the middle of the communications among the participants and is capable of intercepting the messages passed between them. It even has a legitimate identity in the communication systems, of which it can take advantage.

Bearing this major objective, this thesis presents an analytical discussion about the applicability of the supervisory control framework of DESs to the analysis of cryptographic protocols. In order to show the soundness of the proposed method, we conduct case studies with some well-known protocols. All the computational results of this thesis were produced using TCT, a software package for analysis of DESs, designed at the University of Toronto and provided by Professor W. M. Wonham. In addition, we examine the characteristics of the proposed method, such as merits, drawbacks and enhancements. After discussion of the shortcomings of the proposed method, we give some suggestions on how the proposed method may be suitably enhanced. In other words, this thesis not only applies the supervisory control framework of DESs to the analysis of cryptographic protocols, but also provides future directions in this area.

In Chapter 2, we introduce the basic technology of cryptography and the main principles of the supervisory control framework of DESs. Symmetric-key and asymmetric-key systems, digital signature and certificate, and network security are introduced, and automata and languages, supervision and controllability are also discussed.

In Chapter 3, we propose a systematic method for the analysis of attacks on cryptographic protocols, after introducing some basic operations and ideas. In addition, some considerations for establishing finite state automata (FSA) models for the application of the proposed method are discussed. For example, what we call a general "capability model" and general view of an intruder are presented, where the capability model describes a probable way for the intruder to achieve its objectives.

In Chapter 4, we apply the proposed method to four cryptographic protocols. They are the Meyer-Matyas key distribution protocol, the Needham-Schroeder authentication protocol, the Tatebayashi-Matsuzaki-Newman key distribution protocol and Netscape's original Secure Socket Layer (SSL) protocol. The SSL protocol is the most recent protocol in this collection.

In Chapter 5, the shortcoming of the supervisory control framework of DESs for the analysis of cryptographic protocols is discussed and analyzed. Then we introduce and discuss some useful principles for creating legal languages, capability models, and a principle for rapid decision-making about whether or not there is a capability model for certain kinds of messages. Finally, we present an automatic scheme for the proposed cryptographic protocol analysis.

In Chapter 6, we give a summary discussion about cryptographic protocols and their analysis methods. Then we give a review of the contributions of this thesis. Finally, we conclude with suggestions for possible future work.

2 Background

In this chapter we introduce basic concepts and theorems for both cryptography and supervisory control framework of discrete-event systems (DESs). First, the basic concept of cryptography is presented. Then symmetric-key and asymmetric-key cryptographic systems, and digital signature and certificate are addressed. Second, the principal features of the supervisory control framework of DESs are discussed.

2.1 Cryptography

As the Internet and other forms of electronic communication become more prevalent, electronic security is becoming increasingly important. Cryptography is used to protect e-mail messages, credit card information, and corporate data. Cryptography is defined as the technology of protecting information by transforming it into an unreadable format, called *cyphertext* [16,17]. Only those who possess a secret *key* can decrypt the message into plaintext. Encryption and decryption are defined as key-dependent transformations of a message that may be inverted only by using a definite key; the keys used for encryption and decryption are the same or different, depending on the cryptographic algorithm used. Encrypted messages can sometimes be broken by cryptanalysis [16]; however, in this thesis, we assume that the basic cryptographic algorithms are unbreakable. No matter whether the message is delivered by common mail or by electric signal, the basic scenario for message passing is the same and is depicted in Figure 2.1.

The route along which the message travels is recognized as a channel. In order to prevent the leakage of significant information contained in the message, the plaintext message is encrypted into cyphertext first, then it is sent to the receiver. After receiving the cyphertext message, the receiver decrypts it back to plaintext. However, since the channel is open and insecure, interception is possible. The foremost goal of a spy is to violate the secrecy of the communication and benefit from the secret information. More sophisticated goals might be to alter the message so as to confound the receiver with a corrupted message, for instance deceiving the receiver about the identity of the sender.

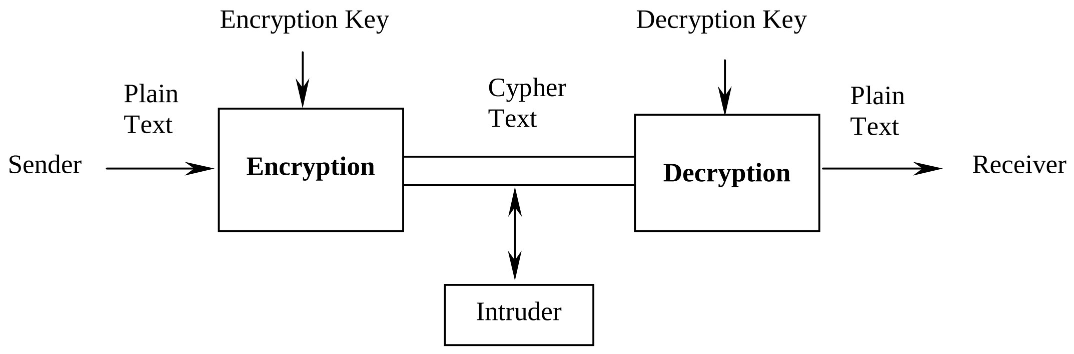

Figure 2.1 Basic view of cryptography

A system is called *computationally secure* if it is secure given limited computational capability of cryptanalysis. A system is called *unconditionally secure* if it can resist any crypt-analytic attack, no matter how much computation is allowed. Shannon discussed two principles to provide a sound theoretical basis for constructing cryptographic algorithms: confusion and diffusion [18]. Confusion uses substitution to mask the plaintext and the key. Diffusion then spreads that effect across the entire cyphertext, thus canceling out any statistical properties of the plaintext. In addition, cryptographic systems can be broadly classified into symmetric-key systems that use a single key that both the sender and recipient have, and asymmetric-key (public-key) systems that use two keys, a public key known to everyone and a private key that only the recipient of messages uses.

Encryption is an atomic operation in cryptography. However, encryption is not synonymous with security, and its improper use can lead to errors. One must be clear about why encryption is being done and its objective. The following list gives some major applications of encryption:

- To obtain confidentiality. In such cases it is assumed that only intended recipients know the key needed to recover a message.
- To guarantee authenticity. In such cases it is assumed that only the proper sender knows the key used to encrypt a message. The encryption clearly contributes to the overall meaning of the message.
- To bind together the parts of a message. Receiving eK(X,Y) is not always the same as receiving eK(X) and eK(Y), where eK(M) indicates that the message is encrypted using a key K.

2.1.1 Symmetric-key and Asymmetric-key Systems

The term “symmetric-key” is used for a cryptographic system in which the sender and receiver of a message share a single, common key to encrypt and decrypt the message.

The most popular symmetric-key system is the Data Encryption Standard. A symmetric-key system is simple and fast, but its main drawback is that the two parties must somehow exchange the key in a secure way. Public-key encryption [17,19] avoids this problem because a public key can be distributed publicly within a certificate, and the private key is never transmitted, assuming that a *public key* is known to everyone and a *secret key* is known only to the recipient of the message. Public key cryptography was invented in 1976 by Diffie and Hellman [19]. It is also called *asymmetric encryption* because it uses two keys instead of one key (*symmetric encryption*), i.e., a public key and a private or secret key. For example, when Alice wants to send a secure message to Bob, she uses Bob's public key to encrypt the message. Bob then uses his private key to decrypt it. In other words, in asymmetric-key cryptographic systems, each user has both a public and a private key, and the two users can communicate knowing only each other's public-key.

In a public-key cryptographic system, let P denote an encryption key and let S denote its corresponding decryption key, then it follows a prerequisite that the keys should be so skillfully constructed that the derivation of the deciphering key S from the enciphering key P is intractable. A task is said to be *intractable* (computationally infeasible) if its cost as measured by either the amount of memory employed or the runtime spent is perhaps finite but prohibitively large. The prerequisite is usually satisfied by means of the difficulty of solving a related inverse problem within a well-defined algebraic structure [20]. A characteristic of public-key cryptographic systems is that the encryption key P can be made public without compromising the secrecy.

2.1.2 Digital Signature and Certificate

Sometimes documents are stored in digital form, and digital signatures may be required. A digital signature is a digital code that can be attached to an electronic message that uniquely identifies the sender. Like a written signature, the purpose of a digital signature is to guarantee that the individual sending the message really is who he or she claims to be. Like any digital information, digital signatures can be copied arbitrarily. This implies that the signatures of the same signer on different messages have to be completely different. Therefore, digital signatures cannot be any sort of digitized version of handwritten signatures. Instead, digital signature schemes are a specially developed class of mathematical functions. Digital signatures usually employ asymmetric-key cryptography. The signature can be verified by anyone using the signer's public verification key. A one-way function is usually employed in digital signature schemes. A function f is a one-way function if, for any argument x in the domain of f, it is easy to compute the corresponding value $f(x)$, yet, for almost all y in the range of f, it is computationally infeasible to solve the equation $y=f(x)$ for any suitable argument x. Digital signatures are especially important for e-commerce and are a key component of most authentication schemes. To be effective, digital signatures must be unforgeable. There are a number of different encryption techniques to guarantee this level of security. A detailed discussion on digital signatures can be found in [21,22,23].

A digital certificate, also known as a Digital ID, is the electronic equivalent of a passport or a business license. It is a credential, issued by a trusted authority, that individuals or organizations can present electronically to prove their identity or their right to access information. When a certification authority issues Digital IDs, it verifies that the owner is not claiming a false identity. Just as when government issues a passport, it is officially vouching for the identity of the holder. In other words, a digital certificate is an electronic "credit card" that establishes the owner’s credentials when doing business or other transactions through computer network. A digital certificate is issued by a certification authority (CA). It contains the owner’s name, a serial number, expiration dates, a copy of the certificate owner's public key (used in successive communications for encrypting and decrypting messages and digital signatures), and the digital signature of the certificate-issuing authority so that a recipient can verify that the certificate is real. The X.509 standard [6] is the most widely used standard for digital certificates; many digital certificates conform to it. Digital certificates can be kept in registries so that the authenticated users can look up other users' public keys. The CA makes its own public key readily available through print publicity or perhaps on the Internet. The most common use of a digital certificate is to verify that a user sending a message is who he or she claims to be, and to provide the receiver with the means to encode a reply. The recipient of an encrypted message uses the CA's public key to decode the digital certificate attached to the message, verifies it as issued by the CA, and then obtains the sender's public key and identification information held within the certificate. With this information, the recipient can send an encrypted reply.

2.1.3 Network Security

In communication networks, an intruder can intercept messages while they are exchanged among entities such as computers, terminals or users. As a result, sensitive information such as passwords or secret documents is at risk of exposure, unless it is protected in some way. It has become clear to both communication suppliers and users that security in communication networks must be addressed. In addition, to communicate securely over insecure channels it is essential that secret keys be distributed securely. Even with asymmetric-key systems, a principal has to know the recipient's public key to encrypt a message for him or her. With all symmetric schemes, each pair of participants who want to communicate securely must have exchanged their own secret key. In the extreme, it might be necessary for the participants to have previously met privately. With asymmetric-key systems, in contrast, only one public key per participant needs to be distributed. As we know, a property characteristic of public-key cryptographic schemes is that the enciphering key can be made public without compromising the secrecy of the deciphering key. So that if a public-key cryptographic scheme is used, the two participants do not have to meet, indeed, they do not have to know each other. If symmetric encryption is used, and there are n users in a system, then $\frac{n(n-1)}{2}$ keys are needed to allow any two to communicate, and every user has to store n-1 keys. This gives a rise to a key agreement protocol. Key agreement protocols typically use symmetric encryption algorithms or variations of the Diffie-Hellman key exchange protocol [19]. The goal of a key agreement protocol is to establish a fresh session key, known only to the participants in the session and possibly some trusted third party, called an Authentication Server (AS). A protocol based on a symmetric-key system assumes that each user has registered a private (secret) key with the *AS*. It is vulnerable to attack because there is a possibility, though very small, for some hacker to log in to the server, then obtain the registered secret keys. In an asymmetric-key system, an agent, say *A*, possesses a public key, denoted P_A, which any other agent can obtain from a key server; also an agent, say *A*, possesses a secret key S_A. The *AS* uses the public keys to protect (by encryption) the session keys transmitted to the users. Actually, in both systems, key distribution protocols are needed so that the users can acquire keys to establish a secure channel. In symmetric-key systems, the users must acquire a shared communication key; in asymmetric-key systems, the users must acquire each other's public keys. The secrecy of certain pieces of information is critical to the functioning of some cryptographic protocols. Even if it is computationally infeasible to break the encryption algorithm used, the entire system is vulnerable if the keys are not securely distributed.

Advances in computer networks have resulted in tremendous distributed computing systems. In these distributed computing systems, it is essential for any two principals to authenticate each other before communicating. Generally speaking, entity authentication mechanisms allow the verification of an entity's claimed identity by another entity. In terms of cryptography, a claim about the identity of an entity becomes a claim about the identity of the originator of a message. Therefore, entity authentication usually consists of message origin authentication and prevention of the re-use of the message. A principal to be authenticated could corroborate his/her identity by demonstrating knowledge of his/her secret signature key, using his/her signature to sign specific data. A digital signature of a message uniquely defines the identity of the originator. Authentication protocols are employed for applications like remote login. A principal announces its identity, e.g., as a user name, then the system issues a challenge and accepts the principal if a proper response is received. In a challenge/response authentication protocol [16,19], a principal, say *A*, sends a message containing a challenge N_A, usually a nonce (random number used only once), to a principal, say *B*. Then *A* expects a response from *B* to its challenge N_A. Principal *B* responds with a message containing $F(X, N_A)$, a suitable function of N_A and further data fields *X*. If *A* can attribute this message to *B*, then *B* has been authenticated. When an entire communication session has to be authenticated, the initial message exchange will serve to establish a shared secret between the two entities. Further messages are then protected by an integrity mechanism employing the shared secret. When guarding against replay messages from an earlier run of a protocol it is common to use nonces as part of a challenge-response exchange. In fact, a nonce is a random number, with less possibility for replication, since every newly generated random number is quite likely to be different from the one generated before, although, in theory, the same random number could be generated again. Two messages consisting of one block of the same data but bound with different nonces should be recognized as different in the analysis of cryptographic protocols. Nonces are exploited not only as proofs of timeliness but also as substitutes for names. Freshness can also be proved by the use of timestamps. Timestamps are appealing because they seem easier to use than random numbers. If a timestamp is used as a kind of nonce, extra care is needed because timestamps may be predictable to a large extent.

E-commerce applications using the Internet to promote business have been proliferating [1,23,24,25]. Consequently, there is growing interest in the design and development of electronic protocols for conducting commercial transactions over the Internet. However, if electronic transactions on the Internet fail to provide the same degree of accountability as real-life transactions, they are liable to be susceptible to dispute. Accountability is the property by which a third party can verify the unique originator of an object or action. If Alice and Bob share a secret key and Bob receives a message encrypted with this key, he can believe that Alice sent this message. But Bob cannot prove this to a third party, unless he is trusted by the third party not to fake a message using the shared key and hold Alice responsible for that message. Without additional assumptions of trust, symmetric-key encryption schemes do not allow us to hold principals accountable for statements. However, such an assumption of trust would not be necessary if Alice were to digitally sign the message, either using her private key, or using some other strong digital signature algorithm which allows signature authentication. A principal who signs a statement could attempt to repudiate his own statement by intentionally compromising his private key. This fraud can be countered in part by notarization, which shows the time a statement was made. However, this solution would require that, in the absence of overriding proof for the time of key compromise, principals be held accountable for any messages signed with their keys till the key compromise is reported to authorities.

In summary, we can list the five fundamental objectives of computer network security as follows:

- **Authenticity—**ensuring that entities sending messages, receiving messages, or accessing systems are who they say they are, and have the privilege to undertake certain actions.
- **Confidentiality—**enabling only the intended recipient to view an encrypted message.
- **Integrity—**guaranteeing that messages have not been altered by another party since they were sent.
- **Accountability—**including two basic kinds of non-repudiations: non-repudiation of origin is intended to protect against the originator's false denial of having sent the message, establishing the source of a message so that the sender cannot later claim that they did not send the message. Non-repudiation of receipt is intended to protect against a recipient's false denial of having received the message, establishing the destination of a message so that the recipient cannot later claim they did not receive the message.
- **Applicability—**ensuring that security systems can be consistently and thoroughly implemented for a wide variety of applications without unduly restricting the ability of individuals or organizations to go about their daily business.

2.2 Supervisory Control Framework of Discrete-Event Systems

Ramadge and Wonham [26,27] established a supervisory control framework of DESs. It is based on automata theory and the theory of formal languages. The behavior of a DES is described by an automaton and its generated language. Such systems furnish useful models for high-level control of complex systems, such as communication and transportation networks [28,29], computer databases [30], manufacturing systems [31], and power generating stations [32].

2.2.1 Automata and Languages

In the theory of formal languages, a finite nonempty set Σ is called an *alphabet*. The elements of Σ are referred to as *letters*. Finite strings of elements of Σ are referred to as *words* or *traces*. The same letter may occur several times in a word. Also the string consisting of zero letters is considered as a word, i.e., the empty word ε. The length of a word ω is the number of letters in ω, where each letter is counted as many times as it occurs. The set of all words over Σ is denoted by Σ^*. Subsets of Σ^* are referred to as formal languages over Σ. In the supervisory control framework, the alphabet Σ represents a set of events. Strings in supervisory control theory correspond to sequences of events. The process to be controlled can be modeled by a five-tuple deterministic finite-state automaton, defined as follows.

Definition 2.1 [Deterministic Finite-State Automaton]

A deterministic finite-state automaton (DFSA), denoted by *G*, is a five-tuple,

$$G = (X, \Sigma, \delta, x_o, X_\mathrm{m})$$

that generates the language $L(G)$ and marks the language $L_m(\mathrm{G})$ where

X = A finite set of states of G

Σ = A set of events associated with the transitions in G

δ = A partial transition function of G, $\delta: X \times \Sigma \rightarrow X$, for each state (perhaps not every state) $x \in X$, at which $\delta(x, \cdot)$ is defined

x_o = The initial state of G

X_m = The subset of X which represents a set of marked states, which are used to distinguish certain sequences generated by G.

The transition function, δ , is often extended from events to traces recursively:

$$\delta(x, \varepsilon) = x$$

And for $t \in \Sigma^*$, $\sigma \in \Sigma$:

$$\delta(x, t\sigma) = \delta\,(\delta(x, t), \sigma)$$

The language generated by G is defined to be

$$L(G) = \{\ t \in \Sigma^* : \delta(x_o, t) \text{ is defined } \}$$

Likewise, a special subset $L_m(G)$ called the *marked language* of G is defined as follows:

$$L_m(G) = \{\ t \in \Sigma^* : \delta(x_o, t) \in X_m\ \}$$

A marked language allows some subset of plant behavior of particular interest – typically, completed tasks – to be identified.

For example, let us consider a car belt alarm system. It can be modeled by the DFSA as shown in Figure 2.2, where X = {0, 1, 2}, 0 stands for initial state, 1 for test state, 2 for alarm state. The event set is Σ = {SensedKeyOn, SensedBeltOn, SensedBeltOff, Sensed5SecondPassed}. The transition function δ is characterized by the set of transitions {(0, SensedKeyOn, 1), (1, SensedBeltOn, 0), (1, SensedBeltOff, 2), (2, SensedBeltOn, 0), (2, Sensed5SecondPassed, 0)}. The initial state is x_o = 0, and is identified by an arrow leading to state 0. The set of marked states is X_m = {0, 2}, identified by arrows leaving state 0 and state 2. The state 2 will sound an alarm, so it is included in the set of marked states. In general, an arrow entering a state identifies the initial state and an arrow leaving a state identifies a marked state.

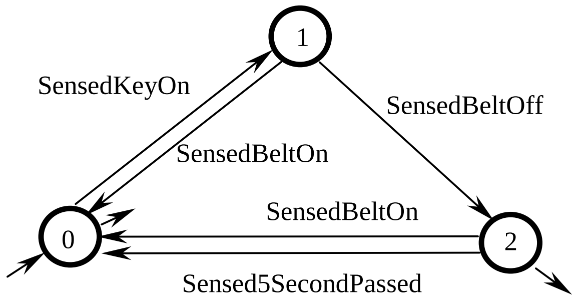

Figure 2.2 A DFSA model for a car belt alarm system

2.2.2 Supervision and Controllability

Supervisory control theory deals with the control of discrete-event systems. It is assumed that some behavior of a plant modeled as a DES is illegal and must be prevented by a controller, called a supervisor [26,27]. Some of the events have a mechanism for their disablement at any time, and the others do not have such a mechanism. Thus the event set Σ is partitioned into two disjoint subsets

$$\Sigma = \Sigma_u \cup \Sigma_c$$

where $\Sigma_u \subseteq \Sigma$ is a set of *uncontrollable* events associated with G, which cannot be directly disabled by control.

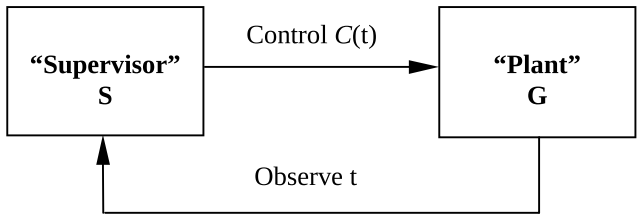

Figure 2.3 Supervision of a DES

The control scheme is shown in Figure 2.3, where the supervisor is a function

$$C: L(G)\rightarrow\{\gamma\in 2^{\Sigma} : \Sigma_u \subseteq \gamma\}$$

As shown in Figure 2.3, after a sequence is generated by the plant G, the C indicates which events should be enabled, subject to the requirement that uncontrollable events cannot be disabled. Let $G = (X, \Sigma, \delta, x_o, X_m)$ and $S = (Q, \Sigma, \xi, q_o, Q_m)$. The generated and marked languages of the closed-loop system under the control C of S are denoted by $L(S/G)$ and $L_{\mathrm{m}}(S/G)=L(S/G)\cap L_{\mathrm{m}}(G)$, respectively. A supervisor S is *complete* with respect to G, if the following is true: for all $s \in \Sigma^*$, $\sigma \in \Sigma$ the three conditions

(1) $s \in L(S/G)$

(2) $s\sigma \in L(G)$ (i.e., $\delta(s\sigma, x_o)$ is defined)

(3) σ is enabled *at* $\xi(s, q_o)$

together imply that $s\sigma \in L(S/G)$, i.e., $\xi(s\sigma, q_o)$ is defined.

Here, we present the definition of prefix-closed language. A string u is a prefix of a string $v\in\Sigma^*$ if for some $w\in\Sigma^*$, $v = uw$. The prefix-closure of a language $L \subseteq \Sigma^*$ is defined as follows:

$$\overline{L} =\{u : uv \in L \text{ for some } v \in \Sigma^*\}$$

The over-bar notation denotes prefix-closure (of languages). We say that L is *prefix-closed* if $\overline{L} = L$.

Consider a sub-language K of $L(G)$, which is the desired set of traces for the controlled discrete-event system. The basic problem in the supervisory control framework of DESs is as follows:

Problem 2.1 [Supervisor Problem]

Given a DES modeled by a DFSA G, $\Sigma_u \subseteq \Sigma$, denoted by G, and a desired language, $K = \overline{K} \subseteq L(G)$, build a complete supervisor S such that $L(S/G) = K$.

In order to discuss how the set of traces may be restricted to the desired subset K, the notion of the controllability of a language with respect to another language is introduced as follows:

Definition 2.2 [Controllable Language]

A language $K \subseteq L(G)$ is controllable with respect to G if

$$\overline{K}\Sigma_u \cap L(G) \subseteq \overline{K}$$

Where, for any languages L and M, the notation LM stands for $\{st \mid s \in L \text{ and } t \in M\}$.

If we interpret $L(G)$ as physically possible behavior and K as legal behavior, an informal description of controllability is that K is controllable if for any sequence of events s that starts out as a legal sequence $(s \in \overline{K})$, the occurrence of an uncontrollable event ($\sigma \in \Sigma_{uc}$)

which is physically possible ($s\sigma \in L(G)$) does not lead the sequence out of the legal range $(s\sigma \in \overline{K})$. According to the work [26,27] of Ramadge and Wonham, we have the following theorem:

Theorem 2.1 [Existence of Supervisor]

Consider a DES, modeled by G. Let $K \subseteq L(G)$ and $K \neq \varnothing$, if K is prefix closed and controllable, then there exists a complete supervisor S such that $L(S/G) = K$.

As mentioned above, the supervisory control framework of DESs draws from automata theory and the theory of formal languages. An automaton model contains not only the language information about a DES, but also the branching structure of allowable sequences of events. It allows the supervisory control problem to be generalized to several process-theoretic semantics. The supervisory control framework of DESs has been used for protocol verification [28,29], it suggests that the supervisory framework of DESs may be applicable to cryptographic protocol verification.

3 Method

In this chapter, we give a brief survey of work related to the analysis of cryptographic protocols. Then we present some basic operations of automata and ideas for establishing the proposed method. In addition, notation and some issues that must be considered for modeling cryptographic protocols and intruders are discussed. Finally, a systematic method for the analysis of cryptographic protocols is proposed.

3.1 Review of Related Work

A protocol is a set of rules or conventions defining an exchange of messages between a set of two or more partners. In cryptographic protocols, all or part of some or all of the messages is encrypted. Cryptographic protocols can help establish secure communication channels in an open environment. It is essential for communicating securely over insecure channels that the related cryptographic protocols be designed properly provided the basic cryptographic algorithms are intractable or secure. However, the design of cryptographic protocols is a notoriously difficult task, and such protocols are themselves subtle, so that a systematic method for the analysis of cryptographic protocols is desirable [7,9,12].

Various techniques and tools have been proposed to analyze cryptographic protocols. For authentication and key distribution protocols, there are some techniques that focus on failures due to message modifications. Meadows [9] gave a survey of the state of the art in the application of formal methods to the analysis of cryptographic protocols. In her paper, she described the most commonly followed approaches to the application of formal methods to cryptographic protocol analysis, i.e., methods based on communicating state machine models, approaches based on a logic of knowledge and belief [33-36], and methods of algebra [37-43]. In particular, some analysis methods are based on the representation of the execution of protocols by finite-state machines (FSMs) [44-47] or by Petri Nets (PNs) [48,49]. These representations lend themselves to algorithmic methods by which a computer can explore systematically the possible sequence of steps of a FSM or a PN. To declare that a protocol is secure, the automatic method must explore the possible sequences of steps of the protocol and verify that all these sequences do not present any security threats. In the case of a complex protocol with a very large number of acceptable sequences of steps, the automatic procedure spends excessive time exploring many sequences before detecting a problematic one. In many cases, this time-consuming search limits the complexity of protocols that can be verified automatically by some existing methods. Thus some papers [9,46] have proposed that automatic procedures should provide an interface for exploiting human experience. However, they have not provided relevant principles for how to incorporate human experience.

3.2 Basic Operations and Ideas

First we introduce some key operations with regard to DFSAs. Unless otherwise specified, these concepts are taken from [26-28,50,51].

Definition 3.1 [Meet Composition]

The meet composition of two automata G_1 and G_2 where

$$G_1 = (X_1, \Sigma_1, \delta_1, x_{1o}, X_{1m})$$

$$G_2 = (X_2, \Sigma_2, \delta_2, x_{2o}, X_{2m})$$

is

$$G_1 \times G_2 = (X_1 \times X_2, \Sigma_1 \cap \Sigma_2, \Delta, (x_{1o}, x_{2o}), X_{1m} \times X_{2m})$$

where,

$$\Delta((x_1, x_2), \sigma) = \begin{cases} (\delta_1(x_1,\sigma), \delta_2(x_2,\sigma)) & if\ \delta_1(x_1,\sigma) \text{ and } \delta_2(x_2,\sigma) \text{ are defined} \\ \text{undefined} & \text{otherwise} \end{cases}$$

A resulting composite state is to be marked if and only if both constituent states are marked. Intuitively, the meet composition machine represents the set of possible actions that are common to both machines G_1 and G_2, hence $L(G_1 \times G_2) = L(G_1) \cap L(G_2)$ and

$L_m(G_1 \times G_2) = L_m(G_1) \cap L_m(G_2)$. The composition defined above is sometimes called "product."

Definition 3.2 [Parallel Composition]

The parallel composition of two automata G_1 and G_2 is

$$G_1 \parallel G_2 = (X_1 \times X_2, \Sigma_1 \cup \Sigma_2, \Delta, (x_{1o}, x_{2o}), \mathrm{X}_{1m} \times \mathrm{X}_{2m})$$

where,

$$\Delta((x_1, x_2), \sigma) = \begin{cases} (\delta_1(x_1,\sigma), \delta_2(x_2,\sigma)) & \text{if } \delta_1(x_1,\sigma) \text{ and } \delta_2(x_2,\sigma) \text{ are defined} \\ (\delta_1(x_1,\sigma), x_2) & \text{if } \delta_1(x_1,\sigma) \text{ is defined and } \sigma \notin \Sigma_2 \\ (x_1, \delta_2(x_2,\sigma)) & \text{if } \delta_2(x_2,\sigma) \text{ is defined and } \sigma \notin \Sigma_1 \\ \textit{undefined} & \text{otherwise} \end{cases}$$

The composition defined above is sometimes called "synchronous product." The parallel composition requires that common events shared by both original DFSAs should occur synchronously. Only if the prerequisite conditions (originating states) in the original DSFAs are met simultaneously can the corresponding transition be defined in the composite DFSA. Those events belonging to either one of the original DFSAs but not belonging to both can occur independently, if they are defined in the corresponding original DFSA. As a matter of fact, there are some other complicated composition operations used in process algebra, for instance, so-called prioritized synchronous composition introduced by Heymann [52], however these are not needed here.

Let us consider an example shown in Figure 3.1.

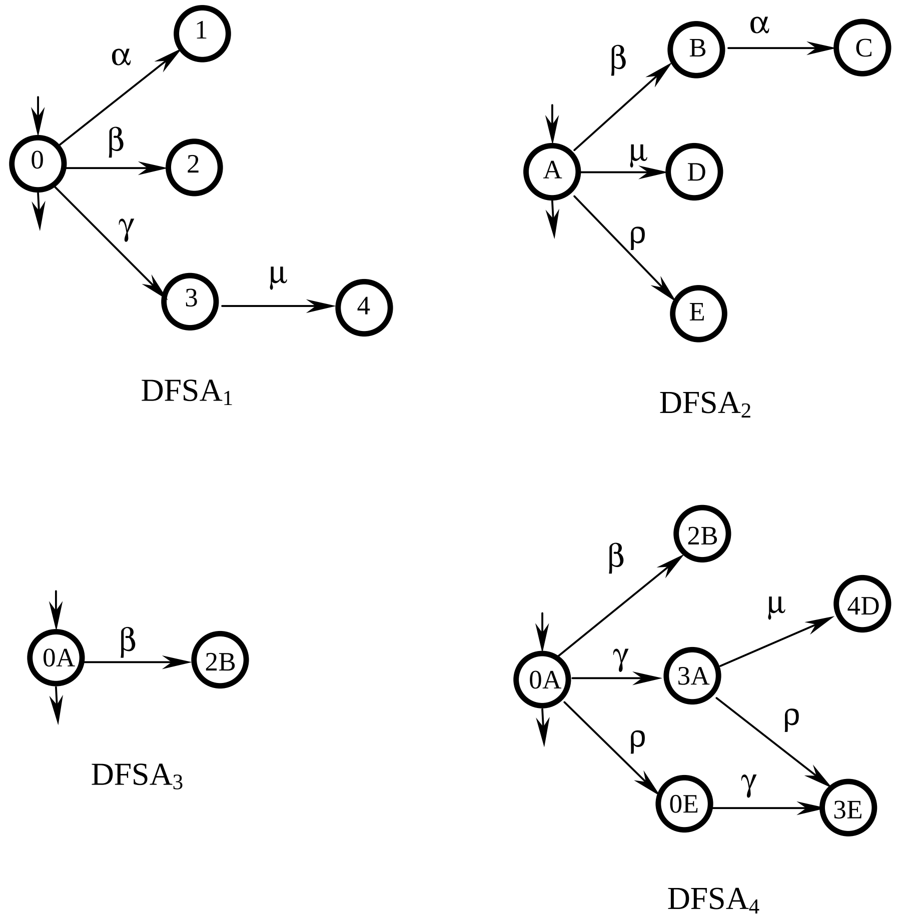

Figure 3.1 An example of the meet composition and parallel composition of two DFSAs
Where DFSA3 = DFSA$_1$ × DFSA$_2$ and DFSA$_4$ = DFSA$_1$ || DFSA$_2$.

The meet composition of DFSA$_1$ and DFSA$_2$ results in DFSA$_3$, i.e., DFSA$_3$ = DFSA$_1$ × DFSA$_2$ and the parallel composition of DFSA$_1$ and DFSA$_2$ results in DFSA$_4$, i.e., DFSA$_4$ = DFSA$_1$ || DFSA$_2$. Though events α, β, and μ occur in both DFSA$_1$ and DFSA$_2$, only β can occur simultaneously in both DFSA$_1$ and DFSA$_2$, thus only β is defined in the resultant automaton DFSA$_3$ from the meet composition of DFSA$_1$ and DFSA$_2$. As for the parallel composition, since γ occurs in only DFSA$_1$, it can occur in the resultant automaton DFSA$_4$ from the parallel composition of DFSA$_1$ and DFSA$_2$. However, after γ occurs, μ can then occur simultaneously in both DFSA$_1$ and DFSA$_2$, thus μ is defined accordingly in the DFSA$_4$.

Always for communication systems, a plant, modeled by a DFSA G, comprises several processes operating concurrently, where each of the constituent processes may be modeled by a five-tuple automaton. We extend the parallel composition from 2 to n, where n is the number of processes under consideration. Let $G_i = (X_i, \Sigma_i, \delta_i, x_{io}, X_{im})$, $i = 1, \ldots, n$ be DFSAs; and let $\Sigma = \cup_i \Sigma_i$, and for each σ let $I(\sigma) = \{i \mid \sigma \in \Sigma_i\}$. Then $G = G_1 \| \ldots \| G_n$, and $G = (X, \Sigma, \Delta, X_o, X_m)$, where $X = X_1 \times \ldots \times X_n$, $X_o = (x_{1o}, \ldots, x_{no})$, $X_m = X_{1m} \times \ldots \times X_{nm}$, and $\Delta(X, \sigma)$ is undefined if $\delta_i(x_i, \sigma)$ is undefined for some $i \in I(\sigma)$; and $X' \in \Delta(X, \sigma)$ with $X' = (x_1', x_2', \ldots, x_n')$ where $x_i' \in \delta_i(x_i, \sigma)$ for all $i \in I(\sigma)$, and $x_i' = x_i$ for $i \notin I(\sigma)$. Thus, in the resultant machine G, events that are common to more than one G_i must occur simultaneously.

In addition, we introduce an operation, called Initial-State Join Composition, as follows:

Definition 3.3 [Initial-State Join Composition]

Consider n DFSAs: $G_i = (X_i, \Sigma_i, \delta_i, x_{io}, x_{im})$, $i = 1, \ldots, n$; $\forall i, j \in [1, n]$ and $i \neq j$, $\Sigma_i \cap \Sigma_j = \emptyset$. Let $\Sigma = \Sigma_1 \cup \Sigma_2 \cup \ldots \cup \Sigma_n$, and introduce the following n DFSAs:

$G_i' = G_i$ with self-loop of $\Sigma \setminus \Sigma_i$ at the initial state of G_i; $i \in [1, n]$.

Then $G = G_1' \times G_2' \times \ldots \times G_n'$ is denoted by $G = G_1 \oplus G_2 \oplus \ldots \oplus G_n$, and is called the Initial-State Join Composition of $G_1, G_2, \ldots, G_n$.

For example, consider two individual DFSAs, denoted by G_1 and G_2, shown in Figure 3.2. Let $G = G_1 \oplus G_2$. By the definitions of the Initial-State Join Composition and Meet Composition (Definition 3.1), we get the DFSA shown in Figure 3.3 for G. From Figures 3.2 and 3.3, we can see that G is formed by joining together both G_1 and G_2 at their initial states.

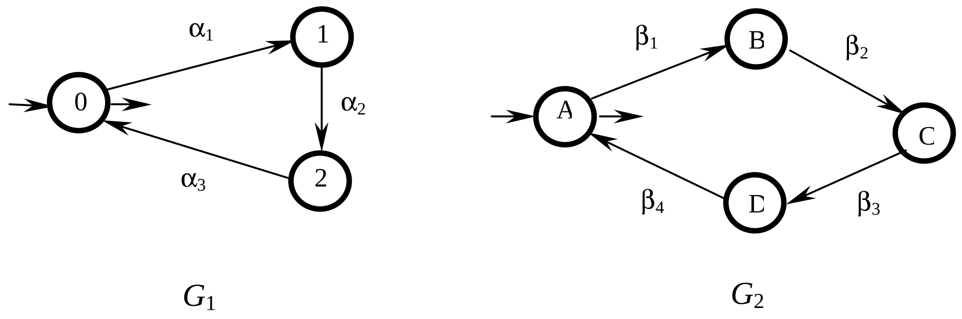

Figure 3.2 Two individual DFSAs

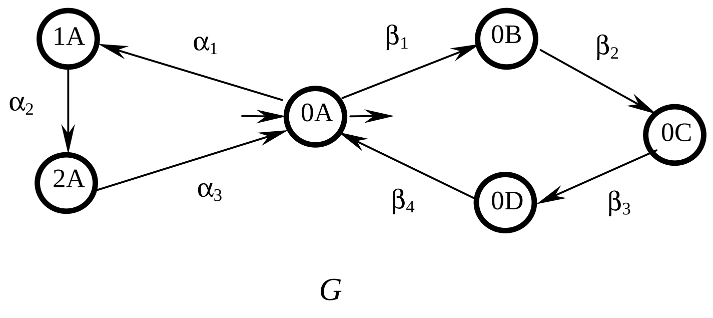

Figure 3.3 The composite DFSA resulting from the Initial-State Join Composition

In order to establish a systematic approach to analyze cryptographic protocols, we cast the problem of cryptographic protocol analysis into whether or not we can construct a controller such that

- The process to be controlled is the behavior process of given entities trying to achieve some secure communication goals, where entities act in accordance with the specification of a cryptographic protocol.
- The objective of the controller is to break the security of the cryptographic protocol.
- The controller is assumed to be able to observe the process, take part in the protocol process, and apply some limited cryptographic operations.

Therefore the analysis of a cryptographic protocol can be reduced to solving a supervisory control problem of DESs, where the legal language K describes a successful attack. If a controller exists, the corresponding cryptographic protocol is considered to be insecure with respect to the attack described by the legal language *K*. In addition, for the application of supervisory control theory to cryptographic protocol analysis, we also have to figure out the following key issues:

- How does one produce a legal language? I.e., what are the relevant sequences of events that describe an intruder's attack of the given protocol?
- Just because a legal language gives us a roadmap that the intruder may use to achieve an attack, it does not guarantee that this attack will be successfully achieved. There will be additional constraints (called *capability model*s) which describe the possible ways for an intruder to acquire messages that have to be sent to other participants, as required by the legal language.

How to build legal language and capability models will be addressed in the next section.

3.3 Legal Language and Capability Model

We introduce the following notation as shown in Table 3.1, for the sake of simplicity, which exploits many conventions from related articles.

Table 3.1 General Notation

Symbol	**Meaning**
X,*Y*	The concatenation of two message *X* and *Y*
e*K*(*X*)	Encryption of message *X* with a key *K*
d*K*(*X*)	Decryption of message *X* with a key *K*

sK(X)	Digital signing of message X with a key K
C$_A$	A certificate of principal A issued by a trusted certification authority. Usually, it contains a public key of its owner and an indicator of the corresponding symmetrical cryptographic scheme, enabling one to verify the identity of the owner via a challenge-response protocol.
vK(X)	Verification of message X with a key K
P$_A$	A public key of principal A
S$_A$	A secret key of principal A
K$_{AB}$	A secret key between principal A and B, usually used as a session key
N$_A$	A nonce produced by principal A. A nonce is a random number generated with the purpose of being used in a single run of a protocol. Nonces are of two kinds: one is predictable and another is not. Sequence numbers and timestamps can be considered as predictable. However, random numbers should be considered as unpredictable. A nonce is used to indicate freshness of its accompanied message, i.e., as an identifier of a protocol run, distinguishing between recent and past.
$A \rightarrow B$:X	A principal A sends message X to a principal B.
$A \rightarrow B$:eP$_B$(X)	A principal A sends to a principal B a message X encrypted with B's public key. So that only B can decrypt it to get the message X. It is assumed that there is a confidential channel in that A can decide that only B be allowed to read the message.
$A \rightarrow B$:eS$_A$(X)	A principal A sends to a principal B a message X encrypted with A's secret key, so that only A can encrypt X using S$_A$. This assumes that there is an authentic channel in that B can decide A must have sent the message X after receiving eS$_A$(X).

Any principal can encrypt a message X using A's public key P_A to produce eP_A(X). Only A can decrypt this message, so that this ensures secrecy. A principal A can sign a message X using its secret key, to produce sS_A(X); any other principal in possession of A's public key can then verify it using A's public key P_A, i.e, vP_A(X). The encryption of a message using A's secret key should assure other agents that this message did really originate from A.

In the interests of simplicity and readability, an event label that refers to a common phrase is written as a single string of characters, which is formed by concatenating all the words or the corresponding abbreviations in the phrase, where a capital letter indicates the beginning of another word. For example, “Send To” and “Receive From” are written as “SendTo” and “RcvFrom,” respectively.

Because what we are studying is whether there are any attacks by an intruder on a given protocol, we focus on the procedures where an intruder is involved in the message communications according to the given protocol. An intruder is assumed to be able to intercept any information traveling in a network, which yields the general view of an intruder as shown in Figure 3.4. By “compose” in Figure 3.4, we mean the intruder transforms some received message into the required message using relevant cryptographic operations discussed in section 2.1.

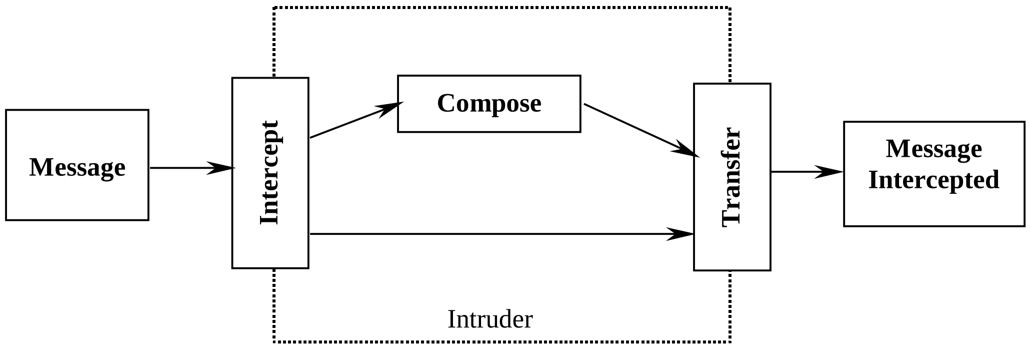

Figure 3.4 General model of an intruder

As shown in Figure 3.4, any message could be intercepted by an intruder, denoted by *I*. For example, let us consider the Meyer-Matyas Protocol [53, Illustrates 11-32], which is designed to distribute authenticated public keys to users who request other users' public keys from a key distribution center (KDC). In this protocol, users and the KDC initially have a pair of secret and public keys as assumed by the Meyer-Matyas Protocol. Only each individual owner knows its own secret key, in addition to knowing the public key of the KDC. Let *S* denote the KDC, *A* and *B* denote an arbitrary pair of users *A* and *B*, where *A* wants to get *B*'s public key through the KDC. Because the intruder can intercept any message exchanged between *S* and *A*, the message sent to *S* by *A* as originally intended may in reality go to the intruder. In what follows, *I* is used to denote an intruder and "*IA*" can be interpreted as "*I* impersonates *A*." Then the basic procedures of the Meyer-Matyas protocol can be described as follows:

A $\rightarrow$ S: {A,B,$eP_S(eS_A(A,N_A))$}, represented by an event

ASendToS{A,B,$eP_S(eS_A(A,N_A))$}.

S $\rightarrow$ A: {A,B,$eP_A(eS_S(A,P_B,N_A))$}, represented by an event

SSendToA{A,B,$eP_A(eS_S(A,P_B,N_A))$}.

They will be changed to the following in consideration of an intruder:

A $\rightarrow$ IS: {A,B,$eP_S(eS_A(A,N_A))$}, represented by an event

ASendToIS{A,B,$eP_S(eS_A(A,N_A))$}.

IA $\rightarrow$ S: {A,B,eP$_S$(eS$_A$(A,N$_A$))}, represented by an event

IASendToS{A,B,eP$_S$(eS$_A$(A,N$_A$))}

The above two steps will be used to replace the original first step of the protocol; also the following two steps will be used instead of the original second step of the protocol.

S$\rightarrow$ IA: {A,B,eP$_A$(eS$_S$(A,P$_B$,N$_A$))}, represented by an event

SSendToIA{A,B,eP$_A$(eS$_S$(A,P$_B$,N$_A$))}.

IS$\rightarrow$ A: {A,B,eP$_A$(eS$_S$(A,P$_B$,N$_A$))}, represented by an event

ISSendToA{A,B,eP$_A$(eS$_S$(A,P$_B$,N$_A$))}.

There is a general procedure for producing a legal language that is used to describe an impersonation attack. Usually, a protocol gives a series of events for participants to follow. Then in the descriptions of those events in the protocol specification, we can use a concatenation of the letter “I” and its original identity character to substitute for the identity of the entity to be impersonated by an intruder. For example “A” is replaced with “IA”. After this replacement, we get the legal language for an impersonation attack.

For instance, let us consider the following authenticating protocol:

A $\rightarrow$ S: {eP$_S$(K$_{AS}$)}, i.e., ASendToS{eP$_S$(K$_{AS}$)}.

S $\rightarrow$ A: {eK$_{AS}$(N$_S$)}, i.e., SSendToA{eK$_{AS}$(N$_S$)}.

A $\rightarrow$ S: {eK$_{AS}$(C$_A$,eS$_A$(N$_S$))}, i.e., ASendToS{eK$_{AS}$(C$_A$,eS$_A$(N$_S$))}.

First, the client *A* sends to the server *S* a session key K_{AS}, encrypted using public key P_S of the server *S*. Then *S* produces a challenge N_S. After receiving N_S, *A* signs and returns it along with a certificate C_A. To study an attack where an intruder impersonates *A* in that protocol, we substitute every "A" with "IA" in the subjective and objective place of every item defining an event. For instances, "ASendToB" is replaced with "IASendToB", the "A" in a subjective place; and "BRcvFromA" is replaced with "BRcvFromIA", the "A" in an objective place. This replacement yields the following procedures:

IA $\rightarrow$ S: $\{eP_S(K_{AS})\}$, i.e., IASendToS$\{eP_S(K_{AS})\}$.

S $\rightarrow$ IA: $\{eK_{AS}(N_S)\}$, i.e., SSendToIA$\{eK_{AS}(N_S)\}$.

IA $\rightarrow$ S: $\{eK_{AS}(C_A,eS_A(N_S))\}$, i.e., IASendToS$\{eK_{AS}(C_A,eS_A(N_S))\}$.

Then, we can build the corresponding DFSA model for the so-called legal language, shown in Figure 3.5, where $\sum_L$ stands for a set of other events involved in the analysis. In this way, the legal language allows for the occurrence of those other events in between the events that are not in the self-loop shown on the Figure 3.5. In fact, it does represent the real situation in that a number of processes may act simultaneously. However, before building up all related DFSA models, we cannot determine what $\sum_L$ should include. When a legal language is used to describe how an intruder could get some secret from other entities, it is necessary for us to determine in what situation the intruder could be able to reach its objective. Though this requirement is very similar to that for describing an impersonation attack, the general procedure for creating such a legal language does not exist. It is rather tricky to obtain this kind of legal language, requiring knowledge of the corresponding cryptographic scheme and experience.

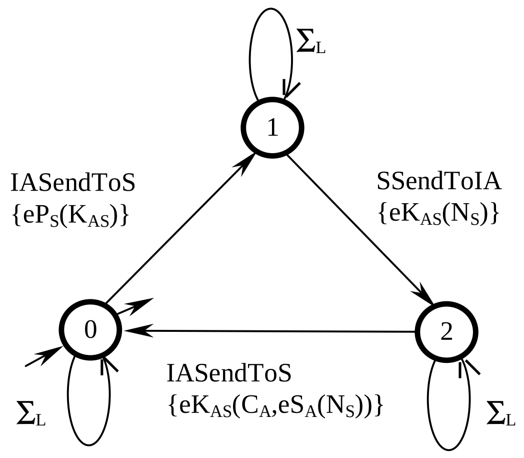

Figure 3.5 An example of legal language *K*
Where the Σ_L is a set of other events to be concerned in analysis

A legal language gives a description of a specific way by which an intruder is going to achieve an attack on a protocol. But such a legal language cannot, and does not, guarantee that an attack described by it is possible. Although the legal language does not guarantee that an attack really exists, the legal language gives us a roadmap along which the intruder may go to realize its objective. If every event shown in the legal language could occur in reality, then the attack follows. This is especially so for those events identifying the intruder's sending some messages. In fact, for each message an intruder must send in the legal language, there will be what we call a *capability model*. For example, in the legal language shown in Figure 3.5, it is necessary for the intruder to send to *S* the message $\{eP_S(K_{AS})\}$ and then the message $\{eK_{AS}(C_A,eS_A(N_S))\}$, so there will be corresponding capability models for them. Before sending an intended message to a target, usually the intruder must compose it first, and therefore the intruder must intercept or obtain some relevant messages from other entities, including from the key distribution center. It is the limited capability that necessitates some prerequisite messages for the intruder to compose a related message. Therefore, we introduce a capability model to describe such a situation, which can be constructed backwards from the corresponding intended message.

The general model of an Intruder's capability for sending a message is shown in Figure 3.6. In order to send the message $\{eP_A(Y)\}$, which is required by the corresponding legal language, the intruder would have to intercept or get some message, such as $\{eP_B(X)\}$, so that he/she can then compose the message $\{eP_A(Y)\}$, then send to *A* the message $\{eP_A(Y)\}$. Of course, it is possible that an intruder has to intercept several kinds of messages to enable the intruder to compose the message required by a legal language. Thus, Figure 3.6 serves only as an indication, not as an exact description. For every message that has to be sent by the intruder, shown on the DFSA for the legal language, there should be a DFSA for the corresponding capability model. If there are a number of messages required by a legal language to be sent by an intruder, it is not convenient for people to create the overall capability model at once. Instead, we try to create a capability model separately for every message required by a legal language to be sent by an intruder, then compose, using the Initial-State Join Composition (Definition 3.3), all those individual capability models to obtain the corresponding overall capability model.

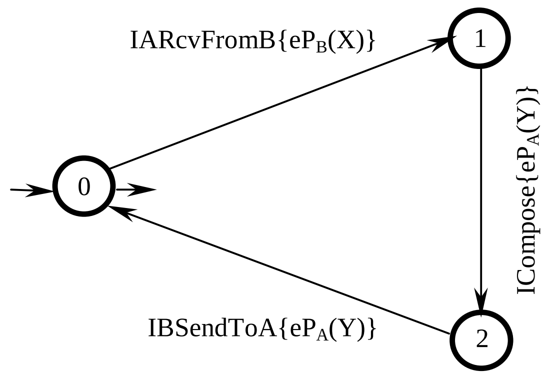

Figure 3.6 The general model of an Intruder's capability where {IARcvFromB{$eP_B(X)$} stands for an event that will allow the intruder to compose $eP_A(Y)$.

3.4 Systematic Steps

In summary, the method proposed in this paper for analysis of a cryptographic protocol can be described as follows:

1) Build a legal language *K* and the corresponding DFSA, which describes what constitutes an intruder's attack on a given protocol.
2) For every message required by the legal language *K* to be sent by the intruder, try to construct the corresponding capability DFSA. An iterative building of capability models for messages required to be sent by the intruder results from the fact that to compose a required message the intruder has to get a set of other messages, which, in turn, requires that the intruder send other messages, and so on. If for any such message it can be **proven** that no corresponding capability DFSA exists, or if after

thorough study, we cannot build a capability model, then an attack identified by the legal language does not exist. Otherwise, if necessary, use the Initial-State Join Composition to build the overall capability DFSA, denoted by P, with alphabet $\sum_P$.

3) Construct DFSAs of entities under study according to the specification of a given protocol, in consideration of the existence of an intruder with defined objective K. That is, those DFSAs should comprise those events which occur in either the legal language or the model of the intruder's capability. In addition, there is a kind of so-called channel model that describes the sending-receiving relationship among related events, because a message cannot be received before it is sent. The principle of cause-effect relationship has to be kept. The composite DSFA of several DFSAs through Parallel Composition is denoted by G, with alphabet $\sum_G$.

4) Build a modified model G' through introducing at every state of the corresponding G self-loops of those events belonging to $\sum_P$ but not belonging to $\sum_G$.

5) Let $\sum=\sum_G\cup\sum_P$. Apply the Supervisory Control Theorem 2.1 in Section 2.2.2 and its supporting software TCT to build S' such that $L(S'/G') = K$. If S' does not exist, then using this capability model, the attack defined by the legal language K on the given protocol does not exist. In that case, go to step 2 and try to establish other capability models with the legal language remaining the same. A different capability model would give another way for the intruder to obtain some different messages if possible, then to compose the messages to be sent by it, and required by the same legal language.

6) If S' exists, let $S'' = S' \times P$, where the capability model P is introduced at this stage to ensure that the attack behaves in accordance with an intruder's abilities. If $S'' = \varnothing$,

then the attack using current capability models does not exist, go to step 2; otherwise go to step 7.

7) In S'' strip off those transitions labelled by events that other participants except for the intruder or intruders incur, namely those events whose labels do not start with "I."

It is worth drawing a flowchart Figure 3.7 for the proposed method, which could give us a clearer image of the proposed method. In addition, in step 2, when we can **prove** that there is no way for an intruder to compose the necessary message, we claim that an attack described by the corresponding legal language does not exist. As shown in Figure 3.7, the proposed method can be characterized as follows: Firstly, we build a legal language. Secondly, we create related capability models according to the legal language. Then we produce corresponding models for related participants and channels based on the capability models and the legal language. It is apparent that this method has the trait of goal-guided back-tracking. Consequently, it efficiently reduces the number of states and transitions under analysis. We could proceed in another way: First, we build models for a set of participants and corresponding channels. Second, we produce a legal language, then try to solve the so-called supervisory control problem. In this case, we would have to take a good number of participants into account so as to make it more likely to find an attack, which results in a huge state-transition space under analysis. Moreover, it would be difficult for us to decide which participants should be taken into consideration before the establishment of a legal language. Clearly, the establishment of a capability model is not an easy task and needs experience.

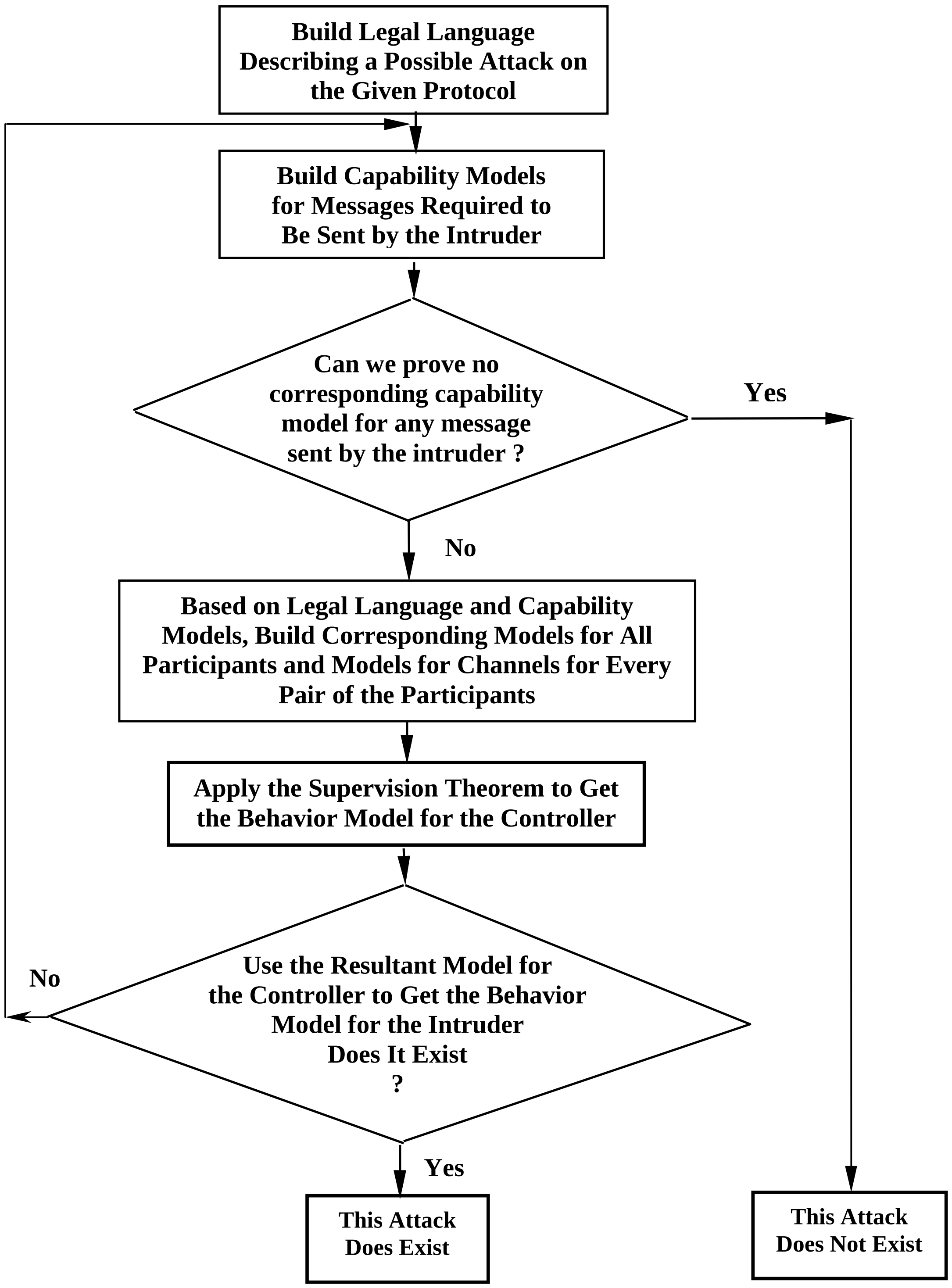
Build Legal Language Describing a Possible Attack on the Given Protocol
Build Capability Models for Messages Required to Be Sent by the Intruder
Can we prove no corresponding capability model for any message sent by the intruder ?
Yes
No
Based on Legal Language and Capability Models, Build Corresponding Models for All Participants and Models for Channels for Every Pair of the Participants
Apply the Supervision Theorem to Get the Behavior Model for the Controller
Use the Resultant Model for the Controller to Get the Behavior Model for the Intruder Does It Exist ?
No
Yes
This Attack Does Exist
This Attack Does Not Exist

Figure 3.7 The flowchart of the proposed method

4 Case Studies

In this chapter, we present some case studies to show the soundness of our proposed method. Those cases include the Meyer-Matyas key distribution protocol, the Needham-Schroeder Authentication Protocol, the Tatebayashi-Matsuzaki-Newman Key Distribution Protocol, and Netscape's original SSL Protocol. The SSL protocol is the most recent protocol. Moreover, the attack presented in this paper on the Tatebayashi-Matsuzaki-Newman key distribution protocol cannot be prevented by the related countermeasure discussed by M. Tatebayashi, N. Matsuzaki and D. B. Newman.

4.1 Analysis of the Meyer-Matyas Key Distribution Protocol

The Meyer-Matyas key distribution protocol (abbreviated MM) is designed to distribute authenticated public keys to users who request other users' public keys from a key distribution center, called a "server" hereafter [53]. In this protocol, users and the server are initially assumed each to have a pair of secret and public keys. Only each individual owner knows his/her own secret key. The public initially knows the server's public key. It is required that the public key endorsed by a user must be the genuine one belonging to him/her when his/her public key is requested by another user. The procedures can be described as follows:

(1) $A \rightarrow S$: A,B,eP_S(eS_A(A,N_A))

(2) $S \rightarrow A$: A,B,eP_A(eS_S(A,P_B,N_A))

The DFSA model for the MM key distribution protocol is sketched in Figure 4.1.

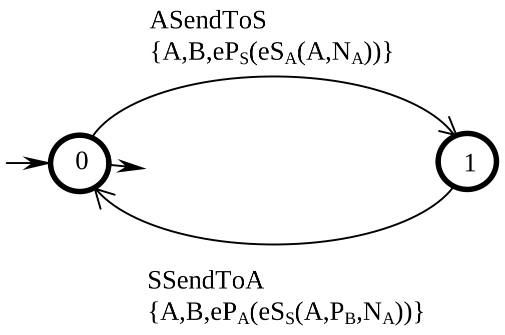

Figure 4.1 A DFSA model for the MM key distribution protocol

In Figure 4.1, A stands for an entity that wants to get the public key of another entity, denoted by B, from the key distribution center, denoted by S. The above shows that the entity A sends to the server S the message {A,B,eP_S(eS_A(A,N_A))}, saying that I, the A, want to get the public key of B, and encloses a challenge (A,N_A) consisting of the identity of A and a nonce N_A. For authenticity the challenge is enciphered using the secret key of A, indicating that the challenge originates from A. Also for secrecy, the authenticated challenge is enciphered using the public key of the server, and only the server can decipher it. Though such measures are employed, the challenge can be broken by an intruder, which will be analyzed in detail below.

For simplicity, some notation is introduced and listed in Table 4.1.

Table 4.1 The basic symbols for the analysis of the MM key distribution protocol

Symbol	**Definition of Events**	**Integer ID for TCT**
α_1	ASendToIS{A,B,eP$_S$(eS$_A$(A,N$_A$))}	2
α_2	ARcvFromIS{A,B,eP$_A$(eS$_S$(A,P$_I$,N$_A$))}	4
β_1	SRcvFromIA{A,I,eP$_S$(eS$_A$(A,N$_A$))}	12
β_2	SSendToIA{A,I,eP$_A$(eS$_S$(A,P$_I$,N$_A$))}	14
γ_1	ISRcvFromA{A,B,eP$_S$(eS$_A$(A,N$_A$))}	20
γ_2	ICompose{A,I,eP$_S$(eS$_S$(A,N$_A$))}	21
γ_3	IASendToS{A,I,eP$_S$(eS$_A$(A,N$_A$))}	23
γ_4	IARcvFromS{A,I,eP$_A$(eS$_S$(A,P$_I$,N$_A$))}	24
γ_5	ICompose{A,B,eP$_A$(eS$_S$(A,P$_I$,N$_A$))}	25
γ_6	ISSendToA{A,B,eP$_A$(eS$_S$(A,P$_I$,N$_A$))}	27
Σ_G	$= \{\alpha_1, \alpha_2, \beta_1, \beta_2\}$	
Σ_P	$= \{\gamma_1, \gamma_2, \gamma_3, \gamma_4, \gamma_5, \gamma_6\}$	
Σ	$= \Sigma_G \cup \Sigma_P$	
Σ_c	$= \{\gamma_2, \gamma_3, \gamma_5, \gamma_6\}$	
Σ_L	$= \Sigma \backslash \{\alpha_1, \gamma_6\}$	

As discussed in Section 3.4, first we have to define the legal language to depict the objective of an assumed intruder. The intruder's purpose here is to try to let *A* endorse the key of the intruder as the public key of *B*, in turn enabling the intruder to impersonate *B* to *A* in a future communication. According to the specification, it is not difficult to construct the legal language *K* as in Figure 4.2. That is, the legal language *K* stands for the objective of the intruder impersonating *B* to *A*. The intruder hopes that P$_I$ could be endorsed by *A* as the public key of *B*, achieved in a legal way in accordance with the specification of the corresponding protocol.

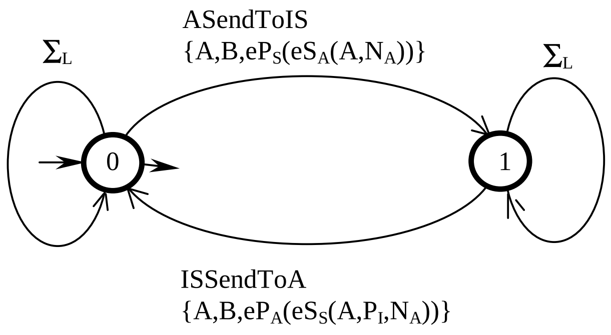

Figure 4.2 The legal language *K* for an attack on the MM key distribution protocol

As shown in the legal language of Figure 4.2, the intruder has to send the message {A,B,eP$_A$(eS$_S${A,P$_I$,N$_A$))}. Bearing in mind the general capability model shown in Figure 3.6, the intruder has to compose the message {A,B,eP$_A$(eS$_S${A,P$_I$,N$_A$))}. However, a possible way for the intruder to accomplish this is to get the message {A,I,eP$_A$(eS$_S$(A,P$_I$,N$_A$))} first, since the difference between the message {A,B,eP$_A$(eS$_S${A,P$_I$,N$_A$))} and the message {A,I,eP$_A$(eS$_S$(A,P$_I$,N$_A$))} is very small in that only the items at the second place of them are different, and these items are not encrypted. Thus follows the capability model P_1 shown in Figure 4.3 for the message {A,B,eP$_A$(eS$_S${A,P$_I$,N$_A$))}.

As shown in Figure 4.3, the intruder has to get the message {A,I,eP$_A$(eS$_S$(A,P$_I$,N$_A$))} from the server *S*, enabling it to compose the required message coming from the legal language. In order to get this message from server *S*, the intruder has to send the message {A,I,eP$_S$(eS$_A$(A,N$_A$))} to the server *S*. Therefore, the intruder must compose it first. For the intruder to achieve this, it must get the message {A,B,$eP_S(eS_A(A,N_A))$} first. So follows another capability model P_2 for the message {A,I,$eP_S(eS_A(A,N_A))$}, that is shown as Figure 4.4.

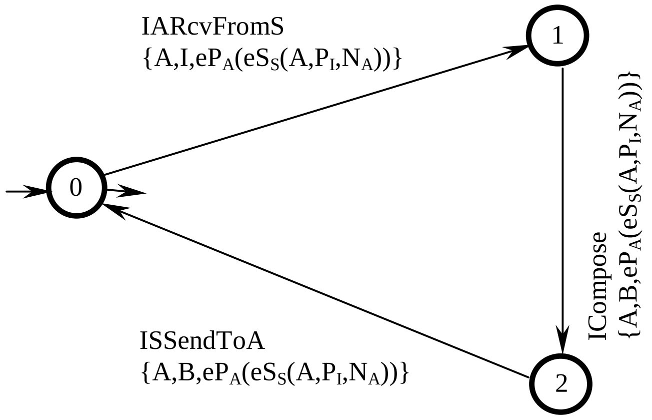
IARcvFromS
{A,I,$eP_A(eS_S(A,P_I,N_A))$}
1
0
ICompose
{A,B,$eP_A(eS_S(A,P_I,N_A))$}
ISSendToA
{A,B,$eP_A(eS_S(A,P_I,N_A))$}
2

Figure 4.3 The capability model P_1 for the message {A,B,$eP_A(eS_S\{A,P_I,N_A))$}

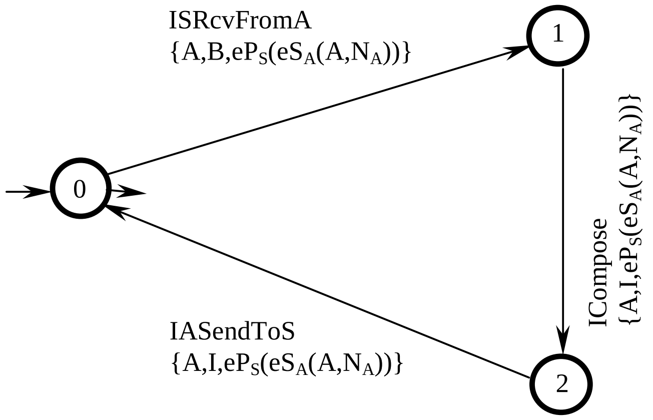
ISRcvFromA
{A,B,$eP_S(eS_A(A,N_A))$}
1
0
ICompose
{A,I,$eP_S(eS_A(A,N_A))$}
IASendToS
{A,I,$eP_S(eS_A(A,N_A))$}
2

Figure 4.4 The capability model P_2 for the message {A,I,$eP_S(eS_A(A,N_A))$}

Thus the overall capability model is

$$P = P_1 \oplus P_2 + \text{Self-Loop of } \Sigma_G \backslash \Sigma_P \text{ at every state}$$

Based on the capability models, we can derive the corresponding models for the initiator denoted by *MA*, the server denoted by *MS*, and the channels *CHNL1* and *CHNL2*. Those models are shown in Figures 4.5-4.8, respectively. For instance, the ISSendToA{A,B,eP$_A$(eS$_S$(A,P$_I$,N$_A$))} indicates that *A* is expected to receive the message {A,B,eP$_A$(eS$_S$(A,P$_I$,N$_A$))}, so there will be an event ARcvFromIS{A,B,eP$_A$(eS$_S$(A,P$_I$,N$_A$))}. In Figure 4.3 and 4.4, there are events IARcvFromS{A,I,eP$_A$(eS$_S$(A,P$_I$,N$_A$))} and IASendToS{A,I,eP$_S$(eS$_A$(A,N$_A$))}, which give us a roadmap for the server *S*. After establishing those models for *A* and *S*, the corresponding channels that govern the sending-receiving relationship can be produced accordingly, where the sending-receiving relationship prevents a message from being received before it is sent.

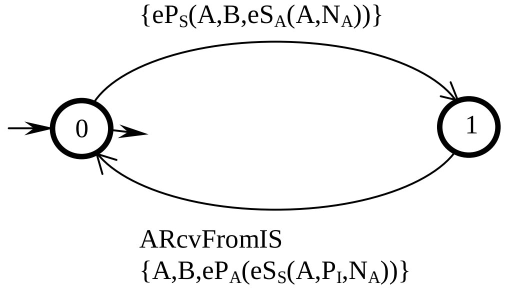
ASendToIS
{eP$_S$(A,B,eS$_A$(A,N$_A$))}
0
1
ARcvFromIS
{A,B,eP$_A$(eS$_S$(A,P$_I$,N$_A$))}

Figure 4.5 The model *MA* for the initiator *A* of the MM protocol

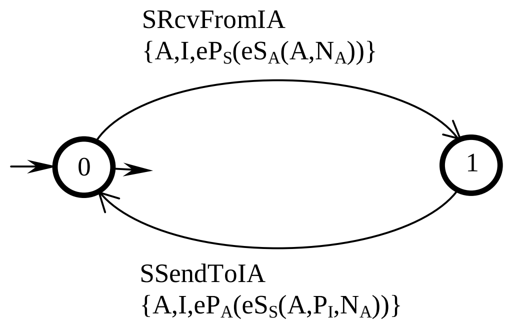
SRcvFromIA
{A,I,eP$_S$(eS$_A$(A,N$_A$))}
0
1
SSendToIA
{A,I,eP$_A$(eS$_S$(A,P$_I$,N$_A$))}

Figure 4.6 The model *MS* for the server *S* of the MM protocol

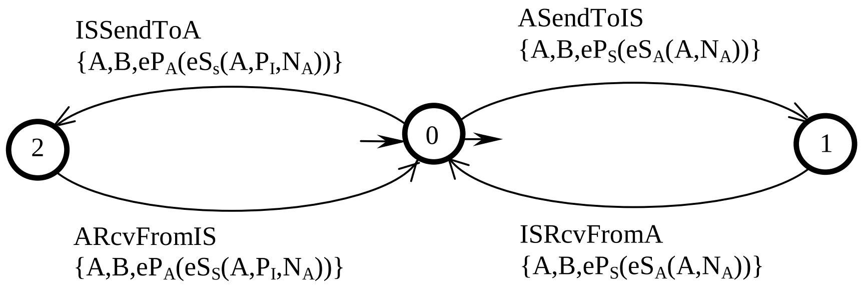
ISSendToA
{A,B,eP$_A$(eS$_s$(A,P$_I$,N$_A$))}
ASendToIS
{A,B,eP$_S$(eS$_A$(A,N$_A$))}
2
0
1
ARcvFromIS
{A,B,eP$_A$(eS$_S$(A,P$_I$,N$_A$))}
ISRcvFromA
{A,B,eP$_S$(eS$_A$(A,N$_A$))}

Figure 4.7 The channel *CHNL1* between the initator *A* and the intruder *IS* of the MM protocol

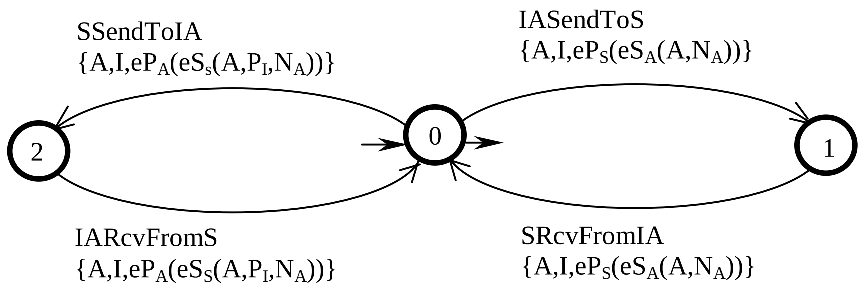
SSendToIA
{A,I,eP$_A$(eS$_s$(A,P$_I$,N$_A$))}
IASendToS
{A,I,eP$_S$(eS$_A$(A,N$_A$))}
2
0
1
IARcvFromS
{A,I,eP$_A$(eS$_S$(A,P$_I$,N$_A$))}
SRcvFromIA
{A,I,eP$_S$(eS$_A$(A,N$_A$))}

Figure 4.8 The channel *CHNL2* between the intruder *IA* and the server *S* of the MM protocol

After application of the Initial-State Join Composition (Definition 3.3 in Section 3.3), we get the overall channel model, denoted by *MC*, i.e., *MC = CHNL1* ⊕ *CHNL2*. Finally, the plant for the analysis of the MM key distribution protocol is given by *G* = *MA* || *MS* || *MC*.

The result produced by the proposed approach and using TCT is shown in Figure 4.9. The result shows exactly what the intruder is supposed to perform. It is not difficult for us to verify the soundness of the result in that the key P_I will be endorsed by *A* as a public key of *B*, which confirms the documented attack [48]. The detailed data are presented in Appendix A.

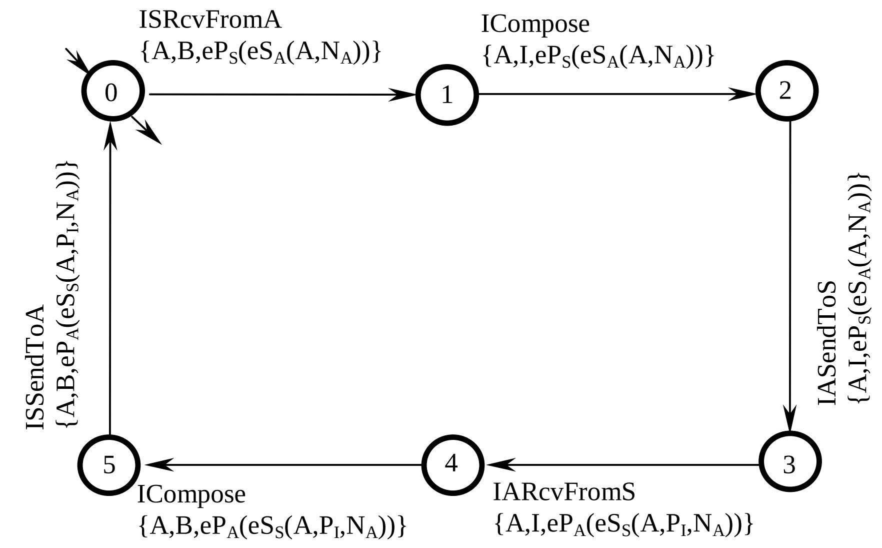

Figure 4.9 The intruder model for an attack on the MM protocol

4.2 Analysis of the Needham-Schroeder Authentication Protocol

In a distributed computer network, it is necessary to have some mechanism whereby a pair of agents can be assured of each other's identity—each should become sure that they really are talking to the other, rather than to an intruder impersonating the other agent. The Needham-Schroeder authentication protocol (abbreviated NS) aims to establish mutual authentication between an initiator *A* and a responder *B* using a public-key cryptographic system [2]. Each principal possesses a public key and a secret key. The public key can be accessed by another principal from a key server. According to the NS authentication protocol, the basic procedures for both the initiator and the responder to authenticate each other are defined as follows:

(1) $A \rightarrow S$: {A,B}

(2) $S \rightarrow A$: $\{eS_S(P_B,B)\}$

(3) $A \rightarrow B$: $\{eP_B(N_A,A)\}$

(4) $B \rightarrow S$: {B,A}

(5) $S \rightarrow B$: $\{eS_S(P_A,A)\}$

(6) $B \rightarrow A$: $\{eP_A(N_A,N_B)\}$

(7) $A \rightarrow B$: $\{eP_B(N_B)\}$

As described above, *A* sends to a server *S* the message {A,B}, saying that I, participant *A*, want to get the public key of *B*. In response, the server *S* sends to *A* the message $\{eS_S(P_B,B)\}$ which consists of the public key of *B*. Because the message is enciphered using the secret key of the server *S*, this indicates the authenticity of the message, as only the server knows its own secret key. After getting the public key P_B of *B*, *A* composes and sends the message $\{eP_B(N_A,A)\}$ to *B*, saying "I am *A* and I want to talk to you, *B*." The challenge N_A enclosed is used to ensure that the upcoming response indeed comes from *B*, since it is assumed that only the *B* can decipher the message $\{eP_B(N_A,A)\}$ to get N_A, and N_A is a nonce. After decrypting the message $\{eP_B(N_A,A)\}$, *B* asks for the public key P_A from the server *S* by sending the message {B,A} to it. After getting P_A, *B* composes and sends the message $\{eP_A(N_A,N_B)\}$ to *A*, presuming that only *A* can decrypt it. Similarly, N_B is a nonce. After receiving and decrypting the message $\{eP_A(N_A,N_B)\}$, *A* responds to *B* by sending the message $\{eP_B(N_B)\}$. Up to then, *A* and *B* have authenticated each other. At first glance, this procedure would function well, but unfortunately, there is an attack on it, which will be analyzed in detail below.

From the description of the protocol, the key steps for participants *A* and *B* to authenticate each other are (3), (6) and (7). That is, *A* sends a challenge $\{eP_B(N_A,A)\}$ in a confidential way, i.e., (N_A,A) is encrypted under the public key of *B* so that only *B* is supposed to be able to get (N_A,A) via decrypting using his/her own secret key. This protocol demands that *B* respond to the key element N_A of the challenge in a simple way that just gives N_A back to the initiator *A*. It is accompanied by a challenge N_B to the initiator *A* from *B*, so as to save communication load. Both N_A and N_B are encrypted using the public key of *A* so that *A* is the only one who is supposed to be able to decrypt it, then read it. Also the response to the challenge N_B of *A* is just to send it back to *B*. Therefore, we can conclude that after the steps shown in Figure 4.10, the participants *A* and *B* will authenticate each other.

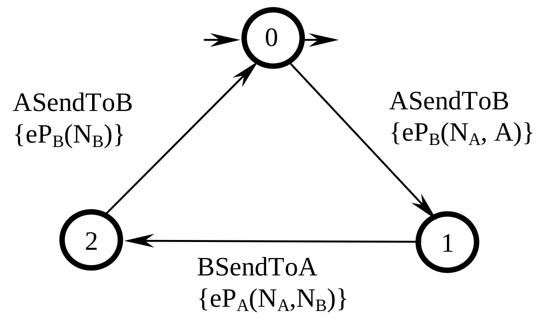

Figure 4.10 The essence of the NS authentication protocol

For simplicity, some symbols are introduced and listed in Table 4.2.

Table 4.2 The basic symbols for the analysis of the Needham-Schroeder Authentication Protocol

Symbol	**Definition of Events**	**Integer ID for TCT**
α_1	ASendToS{A,I}	2
α_2	ARcvFromS{$eS_S(P_I,I)$}	4
α_3	ASendToI{$eP_I(N_A,A)$}	6
α_4	ARcvFromI{$eP_A(N_A,N_B)$}	8
α_5	ASendToI{$eP_I(N_B)$}	10
β_1	BRcvFromIA{$eP_B(N_A,A)$}	22
β_2	BSendToS{B,A}	24
β_3	BRcvFromS{$eS_S(P_A,A)$}	26
β_4	BSendToIA{$eP_A(N_A,N_B)$}	28
β_5	BRcvFromIA{$eP_B(N_B)$}	30
ρ_1	SRcvFromA{A,I}	42
ρ_2	SSendToA{$eS_S(P_I,I)$}	44
ρ_3	SRcvFromB{B,A}	46
ρ_4	SSendToB{$eS_S(P_A,A)$}	48
ρ_5	SRcvFromI{I,B}	50

ρ_6	SSendToI{$eS_S(P_B,B)$}	52
γ_1	IRcvFromA{$eP_I(N_A,A)$}	62
γ_2	ISendToS{I,B}	63
γ_3	IRcvFromS{$eS_S(P_B,B)$}	64
γ_4	ICompose{$eP_B(N_A,A)$}	65
γ_5	IASendToB{$eP_B(N_A,A)$}	67
γ_6	IRcvFromA{$eP_I(N_B)$}	68
γ_7	ICompose{$eP_B(N_B)$}	69
γ_8	IASendToB{$eP_B(N_B)$}	71
γ_9	IARcvFromB{$eP_A(N_A,N_B)$}	72
γ_{10}	ISendToA{$eP_A(N_A,N_B)$}	73
$\sum_G$	$= \{\alpha_i, i=1\ldots5\}\cup\{\beta_i, i=1\ldots5\}\cup\{\gamma_2, \gamma_3, \gamma_4\}$	
$\sum_P$	$= \{\gamma_1, \gamma_5, \gamma_6, \gamma_7, \gamma_8, \gamma_{10}, \gamma_{11}, \gamma_{12}, \gamma_{13}, \gamma_{14}\}$	
$\sum$	$= \sum_G \cup \sum_P$	
$\sum_c$	$= \{\gamma_2, \gamma_4, \gamma_6, \gamma_7, \gamma_9, \gamma_{11}, \gamma_{12}, \gamma_{13}, \gamma_{14}\}$	
$\sum_L$	$= \sum\backslash\{\beta_4, \gamma_6, \gamma_{10}, \gamma_{13}, \gamma_{14}\}$	

According to the proposed analysis scheme, we will start with the establishment of a legal language, capturing an intruder's objective of breaking the NS authentication protocol. The legal language should be a sequence of actions of an intruder, which is supposed to allow it to do the "bad thing" in a feasible way. Here the intruder under investigation is going to impersonate *A* to *B* without being detected. Therefore the events ASendToB{$eP_B(N_A,A)$}, BSendToA{$eP_A(N_A,N_B)$} and ASendToB{$eP_B(N_B)$} in Figure 4.10 should be replaced by IASendToB{$eP_B(N_A,A)$}, BSendToIA{$eP_A(N_A,N_B)$} and IASendToB{$eP_B(N_B)$}, respectively. Consequently, the model shown in Figure 4.11 for the legal language allows the intruder to fulfill his/her objective, i.e., impersonating *A* to *B*. In addition, through comparison, we can find there is a difference between Figure 4.10 and Figure 4.11, in that there are additional self-loops at every state of Figure 4.11.

Because we should allow for occurrences of a set $\sum_L$ of other events between those events shown in Figure 4.10, therefore the self-loops appear in the legal language *K*.

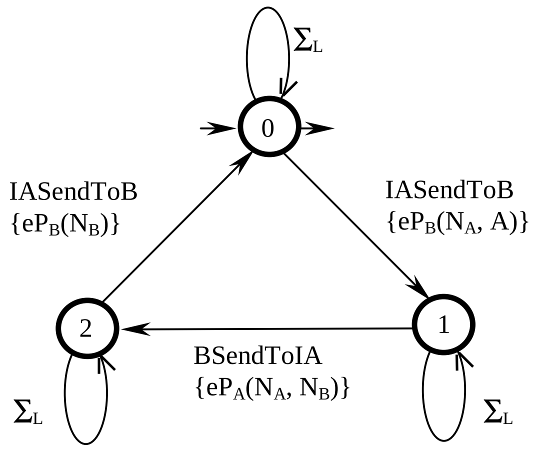

Figure 4.11 The legal language *K* for an attack on the NS authentication protocol

For each message an intruder must send in the legal language, there will be a capability model. According to the legal language, it is necessary for the intruder to send to *B* the message $\{eP_B(N_A,A)\}$ as shown in Figure 4.11 and from the general capability model shown in Figure 3.6, we can produce the capability model P_1 for this message, which is sketched in Figure 4.12.

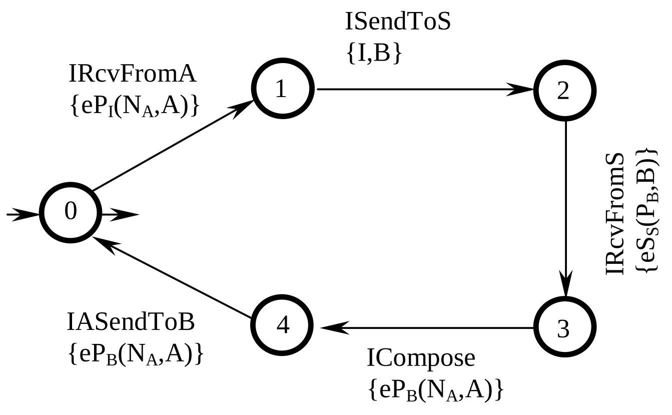

Figure 4.12 The capability model P_1 for the message {eP_B(N_A,A)}

From the legal language shown in Figure 4.11, one can see that the intruder has to send to *B* the message {eP_B(N_B)}; this requires a capability model P_2 for this message, which is sketched in Figure 4.13. Because the intruder has already got the public key P_B of the responder *B* while composing the message {eP_B(N_A,A)}, it is not necessary to ask for the key from the server again.

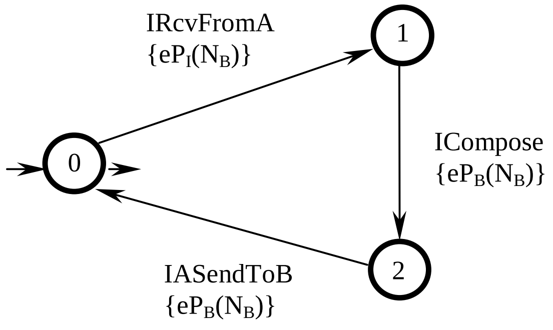

Figure 4.13 The capability model P_2 for the message {eP_B(N_B)}

According to the capability model shown in Figure 4.13 for the message $\{eP_B(N_B)\}$ and the specification of the given protocol, we see that the intruder needs to send to *A* the message $\{eP_A(N_A,N_B)\}$ so as to enable it to receive from *A* the message $\{eP_I(N_B)\}$. Therefore, from the general capability model shown in Figure 3.6, we can produce the capability model P_3 as sketched in Figure 4.14 for the message $\{eP_A(N_A,N_B)\}$.

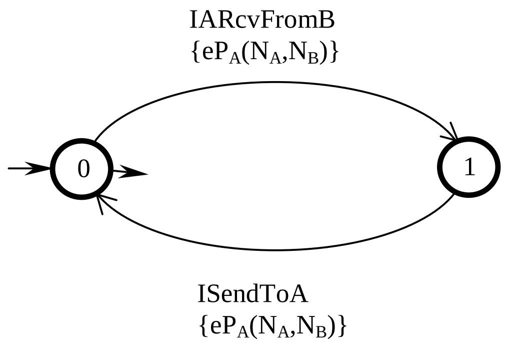

Figure 4.14 The capability model P_3 for the message $\{eP_A(N_A,N_B)\}$

Thus, the overall capability model, denoted by *P*, is as follows:

$$P = P_1 \oplus P_2 \oplus P_3 + \text{Self-Loop of } \Sigma_G \backslash \Sigma_P \text{ at every state}$$

The legal language *K* as shown in Figure 4.11 indicates that *B* functions as a responder with relation to *IA*. Thus, we can derive the models for *A* and *S* according to the specification of the given protocol, in conjunction with Figure 4.12. In addition, the intruder has to send to *B* the message $\{eP_B(N_B)\}$ according to the legal language given in Figure 4.11. In order to achieve this, the intruder wants to get the message $\{eP_I(N_B)\}$, very similar to {$eP_B(N_B)$}, therefore, the intruder should send the message {$eP_A(N_A,N_B)$} to *A* so as to get the corresponding message from *A*. Thus, the DFSA governing the behavior of the initiator *A*, denoted by *MA*, is shown in Figure 4.15. In addition, the intruder sends the message {$eP_B(N_A,A)$} to *B* so as to initiate a communication session in the hope that it is considered by *B* to be *A*. In turn, the intruder is expecting to get the message {$eP_A(N_A,N_B)$} from *B*. Consequently, the intruder has to send the message {$eP_B(N_B)$} to *B* to accomplish the authentication. Therefore, *MB*, the DFSA for modeling the responder *B* follows, and is shown in Figure 4.16. Based on the models for *A* and *B*, it is not difficult to derive the model *MS* for the trusted server, which is depicted in Figure 4.17. In turn, the channel models *CHNL1*, *CHNL2*, *CHNL3*, *CHNL4*, and *CHNL5* for modeling the relationship of sending-receiving can be constructed as in Figures 4.18-4.22, respectively. For the sake of simplicity, the information transferred between the server and another participant is assumed not to be intercepted by the intruder.

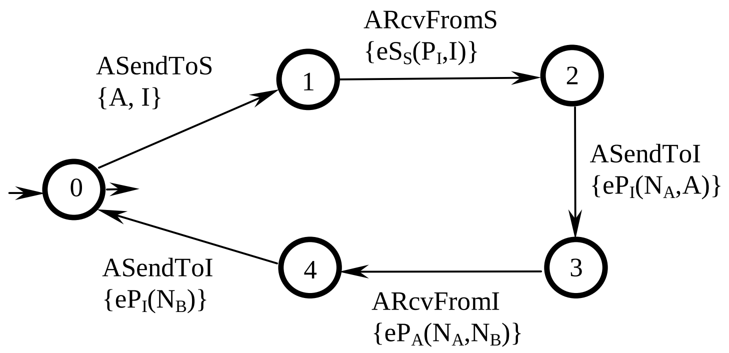

Figure 4.15 The DFSA model *MA* for the initiator *A* of the NS protocol

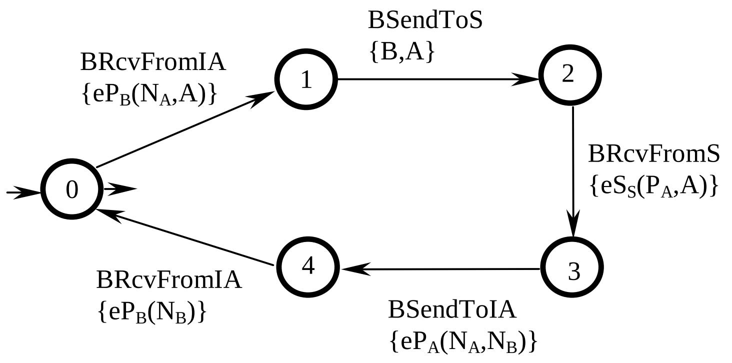

Figure 4.16 The DFSA model *MB* for the responder *B* of the NS protocol

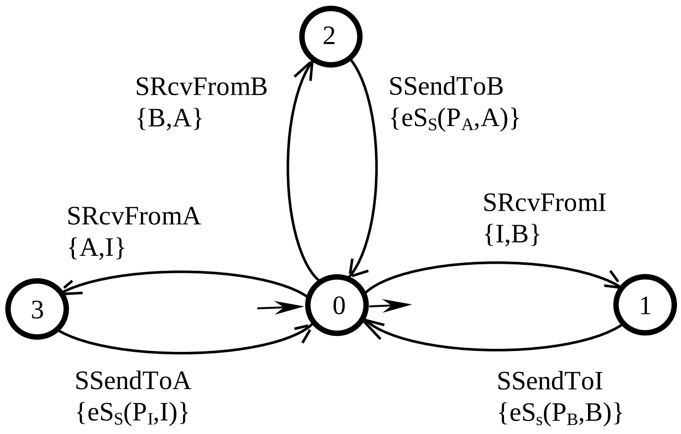

Figure 4.17 The DFSA model *MS* for the server *S* of the NS protocol

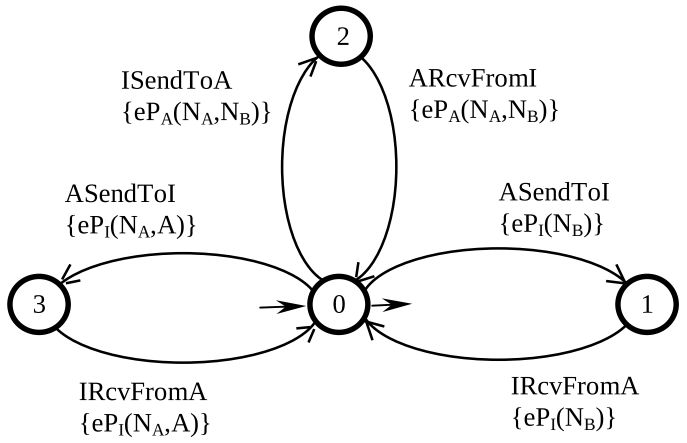

Figure 4.18 The channel *CHNL1* between *A* and *I* of the NS protocol

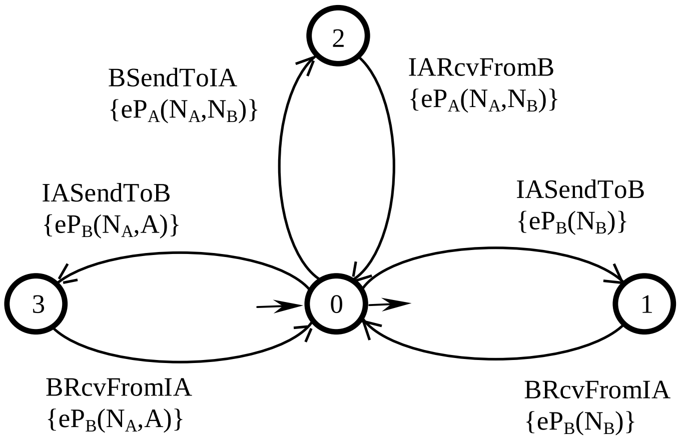

Figure 4.19 The channel *CHNL2* between *B* and *I* of the NS protocol

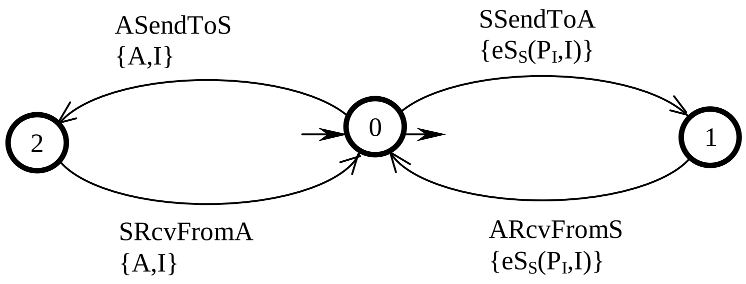

Figure 4.20 The channel *CHNL3* between *A* and *S* of the NS protocol

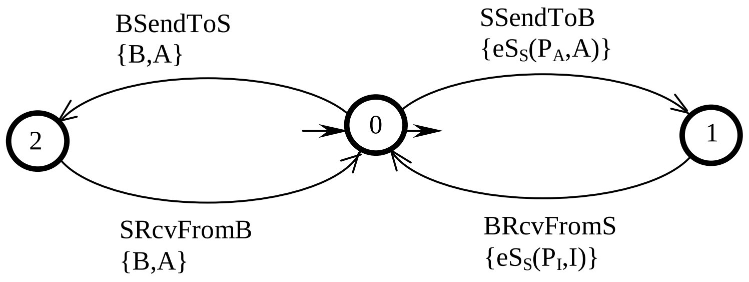

Figure 4.21 The channel *CHNL4* between *B* and *S* of the NS protocol

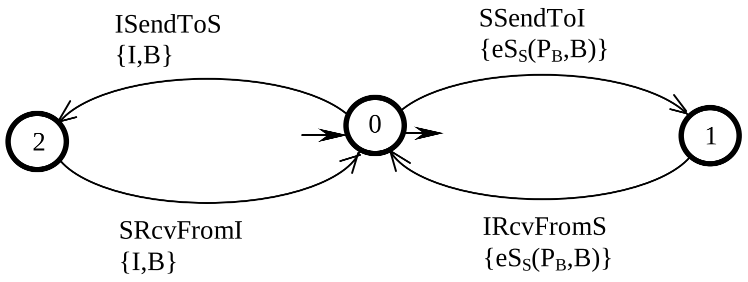

Figure 4.22 The channel *CHNL5* between *I* and *S* of the NS protocol

Thus, the overall channel model *is MC* = *CHNL1* ⊕ *CHNL2* ⊕ *CHNL3* ⊕ *CHNL4* ⊕ *CHNL5*. Finally, the plant under investigation is given by *G* = *MA* || *MB* || *MS* || *MC*.

The result given by TCT shows that an intruder does exist, and its behavior is given in Figure 4.23. The result conforms to the documented attack [5]. The detailed data are shown in Appendix B.

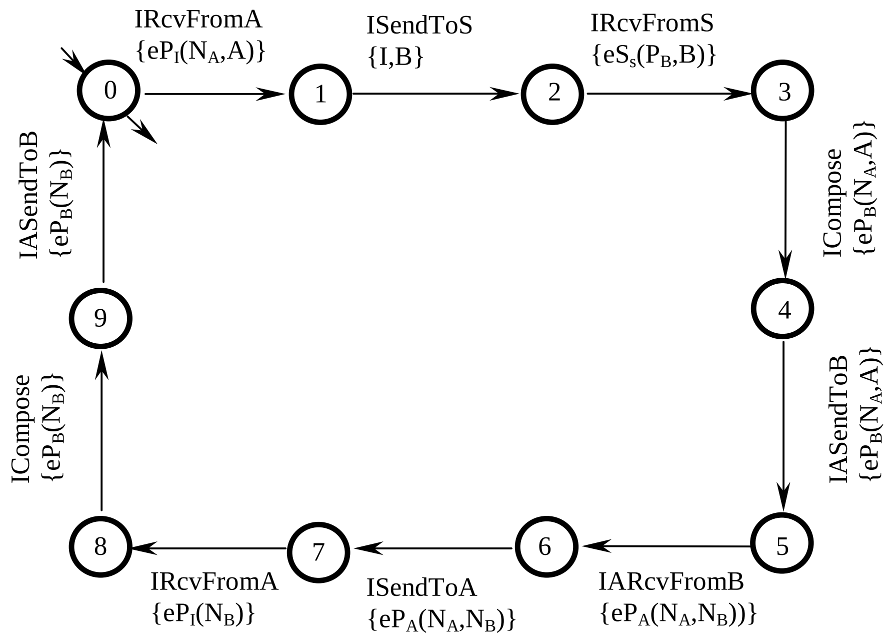

Figure 4.23 The intruder model for an attack on the NS protocol

4.3 Analysis of the Tatebayashi-Matsuzaki-Newman Key Distribution Protocol

Key establishment is a critical issue of data protection while the data are transferred among open networks. M. Takebayashi, N. Matsuzaki and D. B. Newman proposed a key distribution protocol (abbreviated TMN) for digital mobile communication systems of a star-type network [54]. Each user terminal in the system communicates with another user via a network center. The objectives of the proposed key distribution protocol are as follows:

- To remove the key management at a network center
- To enable hardware-limited user terminals to establish a session key in a reasonable time

This protocol assumes that a network has a service center that distributes a session key to the requesting terminals. The TMN protocol uses the public key scheme proposed by R. Rivest, A. Shamir, and L. Adleman [21], the so-called RSA public key scheme, to encrypt a nonce produced at a terminal and decrypt it at a service center. In the classical key distribution method, a symmetric-key cryptographic scheme is used, the session key is encrypted by the terminal's secret key, which requires that the service center should manage each terminal's secret key. If an asymmetric-key cryptographic scheme is employed for securing the session key, then secret key management is shifted to the end user, thus reducing the secret key management problem for the service center. However, the corresponding decryption at a user terminal of limited hardware will take an intolerably long time. Therefore, the TMN protocol uses a simple substitution cryptographic method to secure the session key.

In order to understand the TMN protocol, we hereby briefly describe both RSA and the simple substitution cryptographic schemes. First, we consider the RSA scheme. Let N be a product of two prime numbers p and q. The least common multiplier of p-1 and q-1 is denoted by L. Choose encryption key P_A and the corresponding decryption key S_A such that $P_A S_\mathrm{A}$ = 1 (modulo L). Then, $\mathrm{eP_A}(X)$ = $X^{P_A}(\bmod N) \equiv Y$ and $\mathrm{dS_A}(Y)$ = $Y^{S_A}(\bmod N) \equiv X$. Second, the substitution cryptographic scheme used by the TMN protocol is as follows: eS$_A$(X) = $X + S_A \left(\mathrm{mod}\, N\right) \equiv Y$ and dS$_A$(Y) = $Y - S_A \left(\mathrm{mod}\, N\right) \equiv X$.

In order to tell the difference, we introduce the following notation:

e$_1$P$_A$(X) = $X^{P_A} \left(\mathrm{mod}\, N\right)$

d$_1$S$_A$(X) = $X^{S_A} \left(\mathrm{mod}\, N\right)$

e$_2$S$_A$(X) = $X + S_A (\mathrm{mod}\, N)$

d$_2$S$_A$(X) = $X - S_A (\mathrm{mod}\, N)$

Let *A* denote an initiator, *B* denote a responder, and *S* denote the service center. Thus, the TMN key distribution protocol can be described as follows:

(1) A $\rightarrow$ S: {A,B,e$_1$P$_S$(N$_A$)}

(2) S $\rightarrow$ B: {A,B}

(3) B $\rightarrow$ S: {A,B,e$_1$P$_S$(N$_B$)}

(4) S $\rightarrow$ A: {A,B,e$_2$N$_A$(N$_B$)}

A DFSA model for the TMN key distribution protocol is sketched as Figure 4.24.

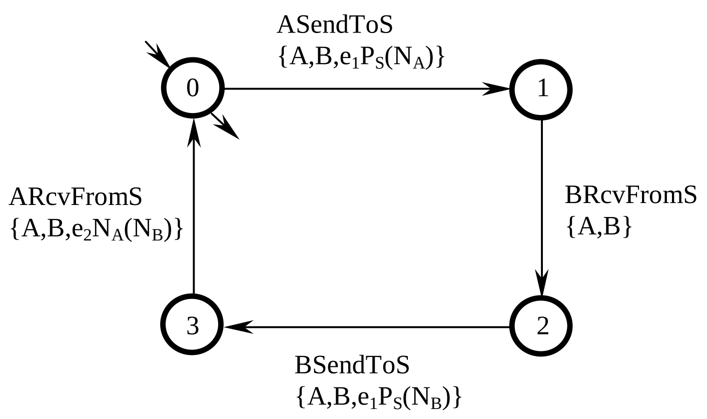

Figure 4.24 A DFSA model for the TMN key distribution protocol

For simplicity, some symbols are introduced and listed in Table 4.3.

Table 4.3 The basic symbols for the analysis of Tatebayashi-Matsuzaki-Newman Key Distribution Protocol

Symbol	**Definition of Events**	**Integer ID for TCT**
α_1	ASendToIS{A,B,$e_1P_S(N_A)$}	2
α_2	ARcvFromIS{A,B,$e_2N_A(N_B)$}	4
β_1	BRcvFromIS{A,B}	12
β_2	BSendToIS{A,B,$e_1P_S(N_B)$}	14
ρ_1	SRcvFromIA{A,B,$e_1P_S(N_A)$}	32
ρ_2	SSendToIB{A,B}	34
ρ_3	SRcvFromIB{A,B,$e_1P_S(N_B)$}	36
ρ_4	SSendToIA{A,B,$e_2N_A(N_B)$}	38
ρ_5	SRcvFromI{I,I′,$e_1P_S(N_B)$}	40
ρ_6	SSendToI′{I,I′}	42
ρ_7	SRcvFromI′{I,I′,$e_1P_S(K)$}	44
ρ_8	SSendToI{I,I′,$e_2K(N_B)$}	46
γ_1	ISendToS{I,I′,$e_1P_S(N_B)$}	61
γ_2	I′RcvFromS{I,I′}	62
γ_3	I′SendToS{I,I′,$e_1P_S(K)$}	63
γ_4	IRcvFromS{I,I′,$e_2K(N_B)$}	64
γ_5	ISRcvFromB{A,B,$e_1P_S(N_B)$}	66
γ_6	ICompose{I,I′,$e_1P_S(N_B)$}	67
γ_7	ISRcvFromA{A,B,$e_1P_S(N_A)$}	68

γ_8	IASendToS{A,B,$e_1P_S(N_A)$}	69
γ_9	IBRcvFromS{A,B}	70
γ_{10}	ISSendToB{A,B}	71
γ_{11}	IBSendToS{A,B,$e_1P_S(N_B)$}	73
γ_{12}	IARcvFromS{A,B,$e_2N_A(N_B)$}	74
γ_{13}	ISSendToA{A,B,$e_2N_A(N_B)$}	75
$\sum_G$	$= \{\alpha_1, \alpha_2\}\cup\{\beta_1, \beta_2\}\cup\{\rho_i, i=1\ldots8\}$ $\cup\{\gamma_2, \gamma_3, \gamma_4, \gamma_7, \gamma_9, \gamma_{12}\}$	
$\sum_P$	$= \{\gamma_1, \gamma_5, \gamma_6\}$	
$\sum$	$= \sum_G \cup \sum_P$	
$\sum_c$	$= \{\gamma_1, \gamma_3, \gamma_6, \gamma_8, \gamma_{10}, \gamma_{11}, \gamma_{13}\}$	
$\sum_L$	$= \sum\backslash\{\gamma_1, \gamma_2, \gamma_3, \gamma_4\}$	

The attack under study on the TMN key distribution protocol is to get a session key used by two participants *A* and *B*. Hereby *I* and *I′* stand for a pair of intruders in cooperation. If *A* is the initiator, the session key is a nonce generated by *B*, denoted by N_B. By the implicit assumption that the RSA algorithm is secure, it follows that intruders cannot derive N_B directly from the knowledge of $e_1P_S(N_B)$. However, if intruders get knowledge of {I,I′,$e_2K(N_B)$}, then by knowing $e_2K(N_B)$ = {K+N_B (mod N)}, they can derive N_B easily, in conjunction with knowing K. According to the specification of the TMN protocol, in order to allow the intruder *I* to receive the message {I,I′,$e_2K(N_B)$}, the intruder *I′* has to send to *S* the message {I,I′,$e_1P_S(K)$}. Before that, however, the event that *I′* receives from *S* the message {I,I′} must occur. Furthermore, the event that *I′* receives from *S* the message must follow the event that *I* sends to *S* the message {I,I′,$e_1P_S(N_B)$}. Therefore, the corresponding legal language *K* is shown as in Figure 4.25.

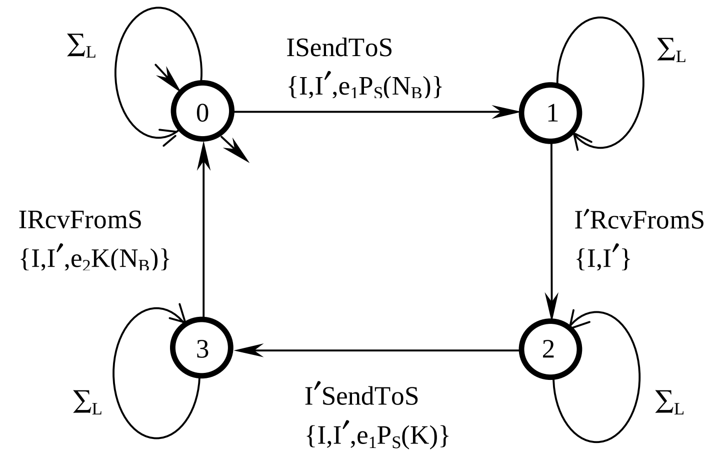

Figure 4.25 The legal language *K* for an attack on the TMN protocol

In addition, before sending to *S* the message $\{I,I',e_1P_S(N_B)\}$, the intruder has to compose this message first. Since an intruder is assumed to be able to intercept any message transmitted among any pair of participants in the network, an intruder can intercept the message $\{A,B,e_1P_S(N_B)\}$. Therefore based on the general capability model shown in Figure 3.6, it follows that the corresponding capability DFSA model *P* for this message is as shown in Figure 4.26.

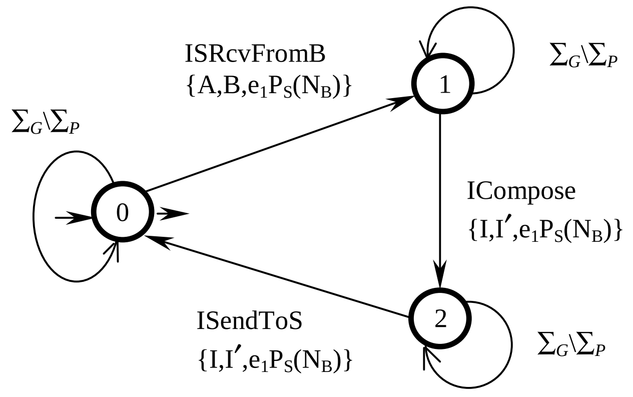

Figure 4.26 The capability model *P* for the message $\{I,I',e_1P_S(N_B)\}$

From those events shown in Figure 4.26 and the discussion above, we should take into consideration a pair of principals, *A* and *B*, in a run of the TMN protocol. Based on the capability models and the specification of the TMN key distribution protocol, we can derive a model for the initiator denoted by *MA*, a model for the responder denoted by *MB*, a model for the server denoted by *MS*, and models for channels *CHNL1*, *CHNL2*, *CHNL3*, *CHNL4*, *CHNL5* and *CHNL6*. Those models are shown in Figures 4.27-4.35, respectively. As we know, the channel models are used to keep sending-receiving orderings among events. No message can be received before it is sent. For the sake of simplicity, we assume that the information transferred between the server and another participant is not intercepted by the intruders.

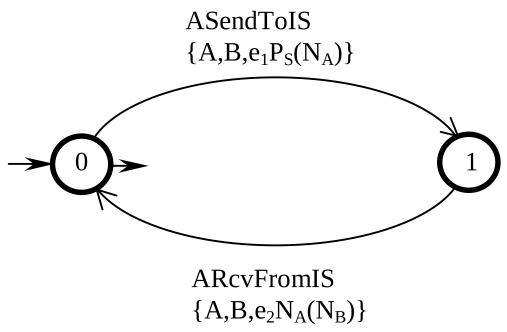

Figure 4.27 The model *MA* for the initiator *A* of the TMN protocol

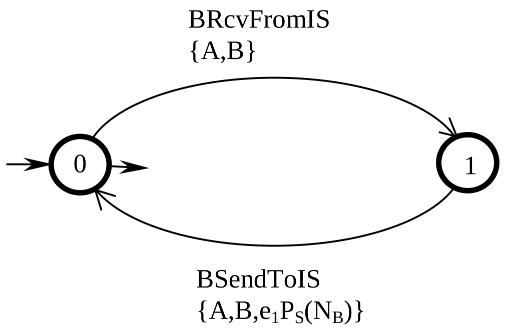

Figure 4.28 The model *MB* for the responder *B* of the TMN protocol

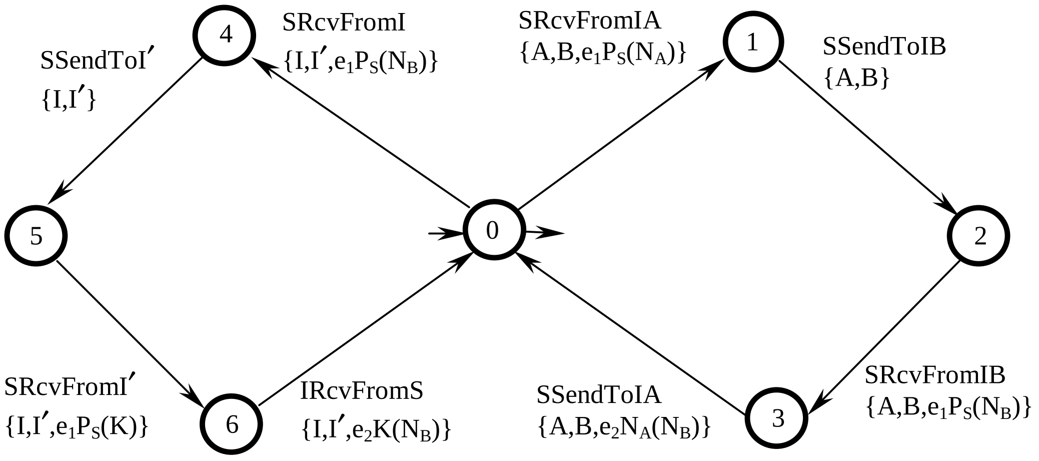

Figure 4.29 The model *MS* for the server *S* of the TMN protocol

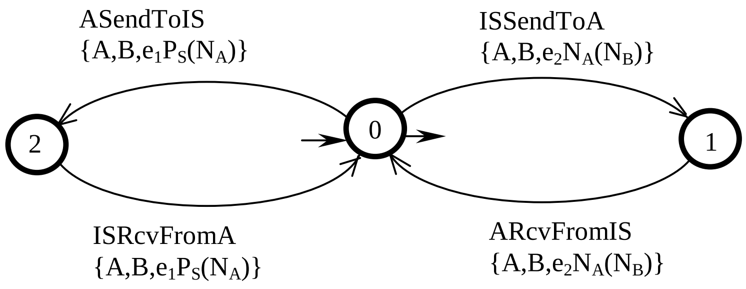

Figure 4.30 The channel *CHNL1* between the intiator *A* and the intruder *IS* of the TMN protocol

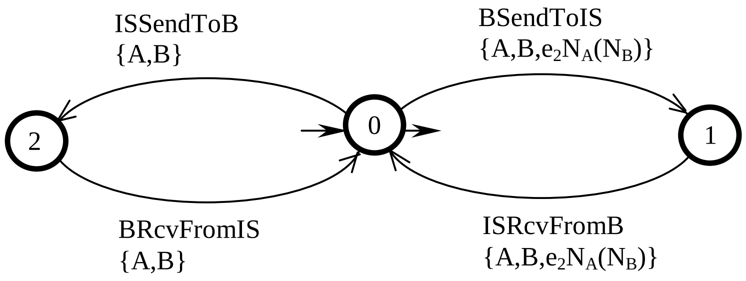

Figure 4.31 The channel *CHNL2* between the responder *B* and the intruder *IS* of the TMN protocol

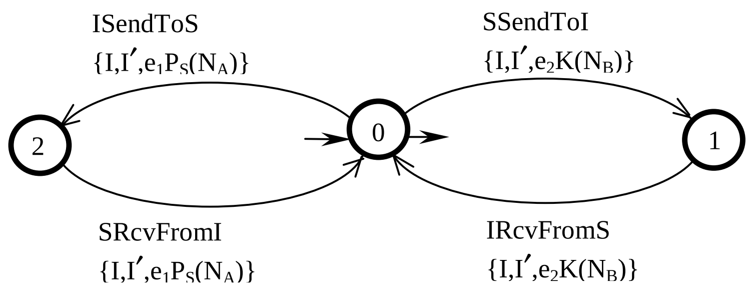

Figure 4.32 The channel *CHNL3* between the server *S* and the intruder *I* of the TMN protocol

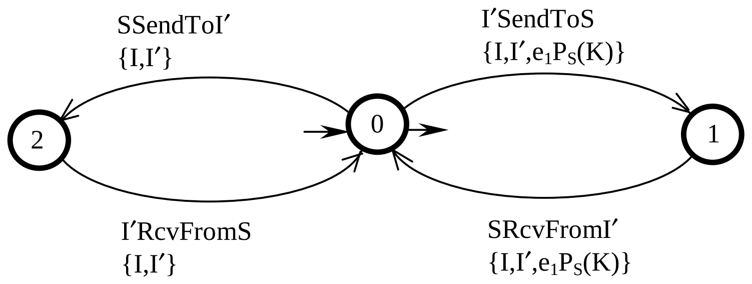

Figure 4.33 The channel *CHNL4* between the server *S* and the intruder *I′* of the TMN protocol

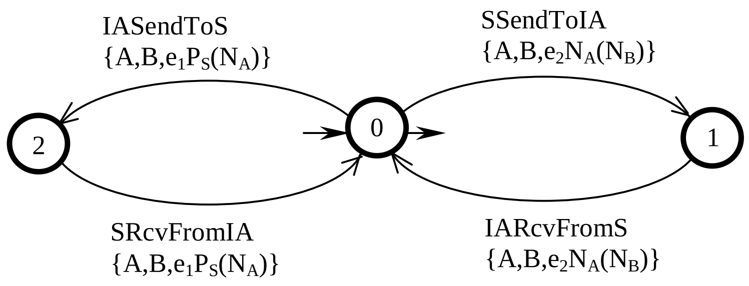

Figure 4.34 The channel *CHNL5* between the server *S* and the intruder *IA* of the TMN protocol

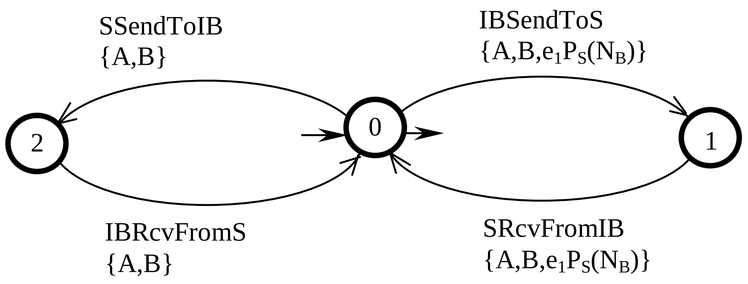

Figure 4.35 The channel *CHNL6* between the intruder *IB* and the server *S* of the TMN protocol

Thus, the overall channel model *is MC* = *CHNL1* ⊕ *CHNL2* ⊕ *CHNL3* ⊕ *CHNL4* ⊕ *CHNL5* ⊕ *CHNL6*. Finally, the plant under investigation is given by *G* = *MA* || *MB* || *MS* || *MC*.

The result given by TCT shows that an intruder does exist; the behavior of the intruder is described by a DFSA that has 84 states and 256 transitions. Though it is hard to depict in a simple figure, it can be verified that the result satisfies the constraints imposed by the legal language and the capability model. This can be done by checking if both the legal language and the capability model are controllable (Definition 2.2 in Section 2.2) with respect to the plant resulting from the parallel composition of the models for the initiator, the responder, the server, the intruder and the channels. The related detailed data are shown in the Appendix C.

In addition, the attack described here cannot be prevented by the countermeasure introduced by M. Takebayyashi, N. Matsuzaki, and D. B. Newman [54]. As we described, in essence, the intruder does not need to find another number, say N_I, to construct $\{I,I',e_1P_S(N_IN_B)\}$ as described in [54]. Instead, the intruder just sends to the server $\{I,I',e_1P_S(N_B)\}$, leaving no way for the server to check whether or not the received number is larger than the prescribed upper bound [54].

4.4 Analysis of Netscape's Original SSL Protocol

Netscape Communications Corporation developed a Secure Sockets Layer protocol (SSL) in 1994 [55]. It is used to transmit private documents via the Internet. The SSL protocol creates a secure connection between a client and a server, establishing a corresponding secret session key. Then both the server and the client use the secret session key to encrypt data that are transferred over the SSL connection. Both Netscape Navigator and Internet Explorer support SSL. Many web sites use the protocol to obtain confidential user information, such as credit card numbers. By convention, web pages that require an SSL connection start with https instead of http. In order to understand the protocol, it is necessary to learn about digital certificates.

The related procedures for the client authentication in the SSL protocol are as follows:

(1) $A \rightarrow S: \{eP_S(K_{AS})\}$

(2) $S \rightarrow A: \{eK_{AS}(N_S)\}$

(3) $A \rightarrow S: \{eK_{AS}(C_A,eS_A(N_S))\}$

First, the client *A* sends to the server *S* a session key K_{AS}, encrypted using the public key P_S of the server *S*. It is worthwhile to point out that K_{AS} is picked by *A*, usually by means of a Random Number Generator. Also K_{AS} is not assumed to be of a pattern word indicating the identity of a generator. Then *S* produces a challenge N_S, which *A* signs and returns along with a certificate C_A. The protocol appears to work well in that it follows the principle of so-called signature-then-encryption. However, there is a flaw in the client authentication with the SSL protocol, as mentioned by Abadi and Needham in [56]. While [56] states that there is a flaw, it does not give a detailed description of how an intruder attacks the protocol. Here, we use the proposed method to get a model that describes in detail how the intruder behaves. According to the description of the SSL protocol, we can draw a diagram, which consists of the procedures for client authentication, shown in Figure 4.36.

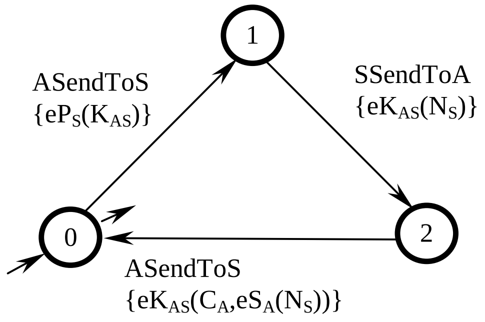

Figure 4.36 The essence of the client authentication in SSL protocol

For simplicity, some symbols are introduced and listed in Table 4.4.

Table 4.4 The basic symbols for the analysis of Netscape's original SSL protocol

Symbol	**Definition of Events**	**Integer ID for TCT**
α_1	AsendToI{$eP_I(K_{AI})$}	2
α_2	ArcvFromI{$eK_{AI}(N_S)$}	4
α_3	AsendToI{$eK_{AI}(C_A,eS_A(N_S))$}	6
β_1	SRcvFromIA{$eP_S(K_{AS})$}	12
β_2	SsendToIA{$eK_{AS}(N_S)$}	14
β_3	SRcvFromIA{$eK_{AS}(C_A,eS_A(N_S)$}	16
γ_1	IASendToS{$eP_S(K_{AS})$}	41
γ_2	IrcvFromA{$eK_{AI}(C_A,eS_A(N_S))$}	42
γ_3	Icompose{$eK_{AS}(C_A,eS_A(N_S))$}	43
γ_4	IASendToS{$eK_{AS}(C_A,eS_A(N_S))$}	45
γ_5	IrcvFromA{$eP_I(K_{AI})$}	46
γ_6	IARcvFromS{$eK_{AS}(N_S)$}	48
γ_7	Icompose{$eK_{AI}(N_S)$}	49
γ_8	IsendToA{$eK_{AI}(N_S)$}	51
$\sum_G$	$= \{\alpha_i, i=1\ldots3\} \cup \{\beta_i, i=1\ldots3\} \cup \{\gamma_1\}$	
$\sum_P$	$= \{\gamma_2, \gamma_3, \gamma_4, \gamma_5, \gamma_6, \gamma_7, \gamma_8\}$	
$\sum$	$= \sum_G \cup \sum_P$	
$\sum_c$	$= \{\gamma_1, \gamma_3, \gamma_4, \gamma_7, \gamma_8\}$	
$\sum_L$	$= \sum \backslash \{\beta_2, \gamma_1, \gamma_4\}$	

Here, we study the attack of an intruder's impersonating *A* in that it acts like *A*, thus we substitute every A with IA in the event labels (but not in the message contents) in Figure 4.36. This yields the legal language *K* as shown in Figure 4.37.

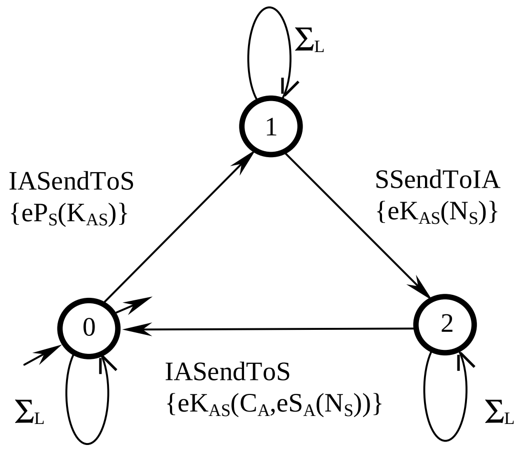

Figure 4.37 The legal language *K* for an attack on the SSL protocol

According to the legal language shown in Figure 4.37, it is necessary for the intruder to send to *S* the message $\{eP_S(K_{AS})\}$. It is easy for an intruder to compose this message in that K_{AS} is a normal random number, so the intruder can just pick some number, then encrypt it using the public key P_S of the server. The intruder can now claim that the picked number is K_{AS}, since the server would not be able to tell the difference. Therefore we don't need to build a capability model for $\{eP_S(K_{AS})\}$. For the message $\{eK_{AS}(C_A,eS_A(N_S))\}$, there will be a capability model. If the intruder, denoted by *I*, can get a message $\{eK_{AI}(C_A,eS_A(N_S))\}$ where K_{AI} is a session key between the client *A* and the intruder *I*, then the intruder can compose the required message$\{eK_{AS}(C_A,eS_A(N_S))\}$from the message $\{eK_{AI}(C_A,eS_A(N_S))\}$ since the intruder knows both the K_{AI} chosen by the client *A* and K_{AS} chosen by itself. Referring to the general capability model shown in Figure 3.6, we can produce the capability model P_1 for this message as shown in Figure 4.38.

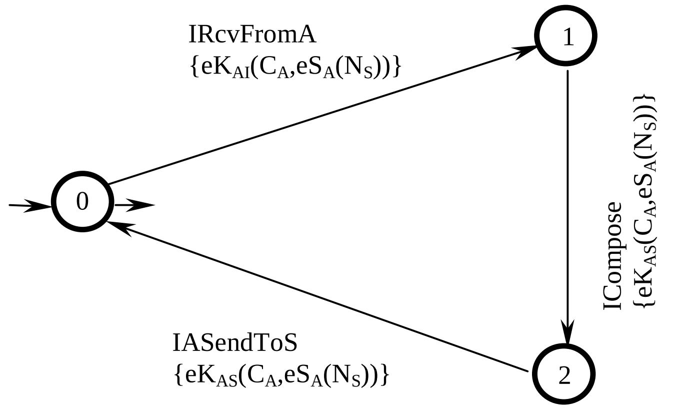
IRcvFromA
{eK$_{AI}$(C$_A$,eS$_A$(N$_S$))}
1
0
ICompose
{eK$_{AS}$(C$_A$,eS$_A$(N$_S$))}
IASendToS
{eK$_{AS}$(C$_A$,eS$_A$(N$_S$))}
2

Figure 4.38 The capability model P_1 for the message eK$_{AS}$(C$_A$,eS$_A$(N$_S$))

From Figure 4.38, we see that the event IRcvFromA{eK$_{AI}$(C$_A$,eS$_A$(N$_S$))} is a possible event and so we have to figure out how the event IRcvFromA{eK$_{AI}$(C$_A$,eS$_A$(N$_S$))} could occur. Studying the specification of the SSL protocol, we can find one way where if the intruder first sends to *A* the message {eK$_{AI}$(N$_S$)}, then the event will follow. In this case, we would need a capability model P_2 for the message {eK$_{AI}$(N$_S$)}, which is sketched in Figure 4.39.

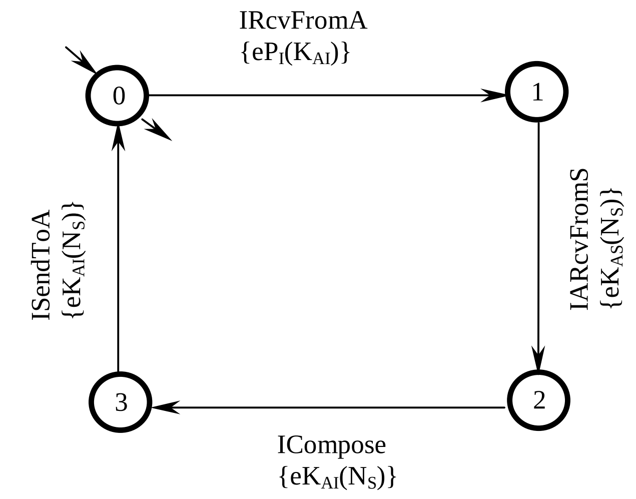

Figure 4.39 The capability model P_2 for the message {eK_{AI}(N_S)}

Thus, the overall capability model, denoted by *P*, is as follows:

$$P = P_1 \oplus P_2 + \text{Self-Loop of } \Sigma_G \setminus \Sigma_P \text{ at every state}$$

Based on the capability models and the specification of the SSL protocol, we can derive a model for the initiator, denoted by *MA*, a model for the server, denoted by *MS*, and models for the channels *CHNL1* and *CHNL2*. They are shown in Figures 4.40-4.43. For example, since the intruder expects to receive from *A* messages {eP_I(K_{AI})} and {eK_{AI}(C_A, eS_A(N_S))}, it follows that *A* should initiate a secret session with the intruder as a server. Therefore *A* is supposed to act as shown in Figure 4.40. It is rather apparent that the behavior of the server *S* is governed by the model shown in Figure 4.41. In order to prevent a message from being received before it is sent, we introduce a channel model to guarantee this cause-effect relationship. After the establishment of DFSA models for *A* and *S*, we can build corresponding DFSA models for the relevant channels. For instance, in the model shown in Figure 4.42 for a channel between *A* and the intruder, the event IRcvFromA{$eP_I(K_{AI})$} must follow the event ASendToI{$eP_I(K_{AI})$}. Similarly, the event ARcvFromI{$eK_{AI}(N_S)$} should occur after the occurrence of the event ISendToA{eK_{AI}(NS)}. By the Initial-State Join Composition, we get the overall channel model, i.e., $MC = CHNL1 \oplus CHNL2$. Finally, the plant under investigation is given by $G = MA \parallel MS \parallel MC$.

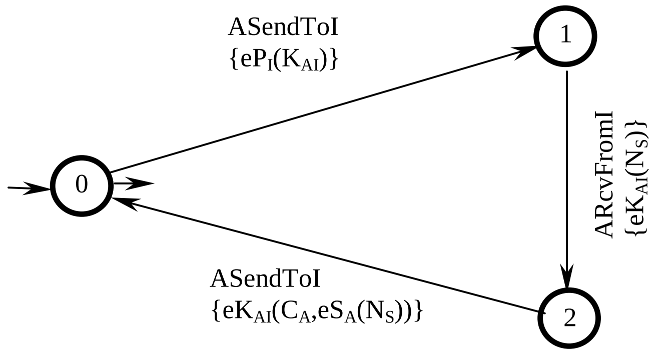

Figure 4.40 The model *MA* for the initiator *A* of the SSL protocol

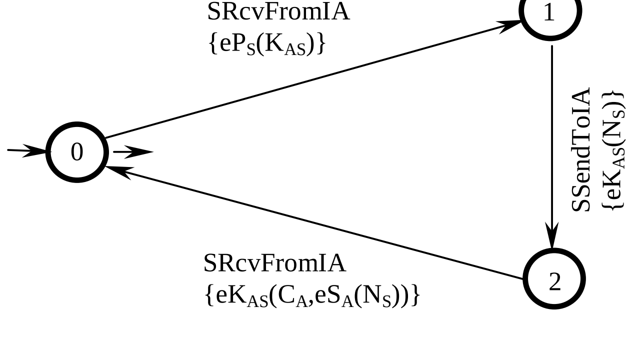

Figure 4.41 The model *MS* for the server *S* of the SSL protocol

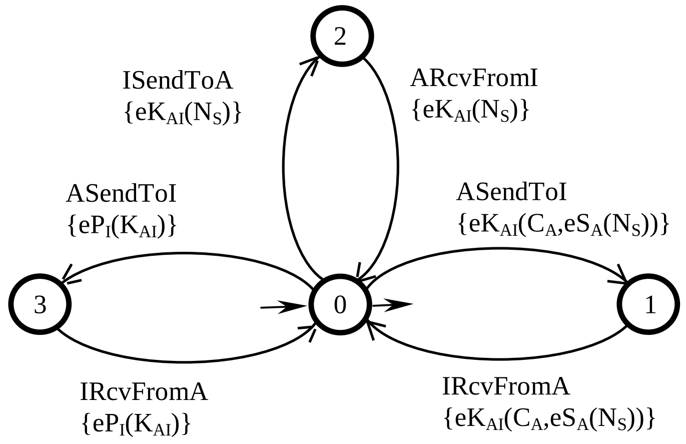

Figure 4.42 The channel *CHNL1* between *A* and *I* of the SSL protocol

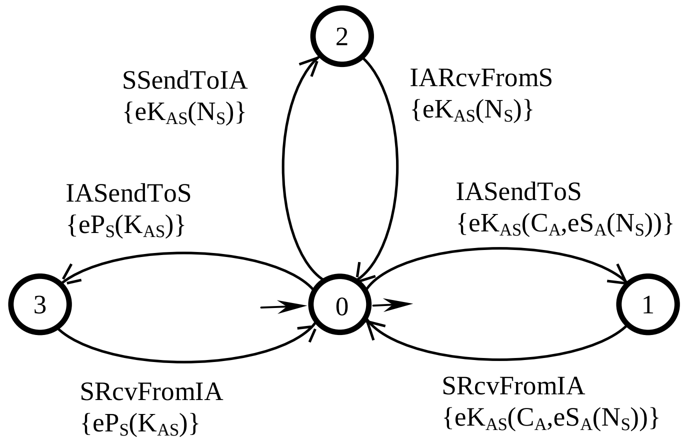

Figure 4.43 The channel *CHNL2* between *IA* and *S* of the SSL protocol

The result given by TCT shows that an intruder does exist; the behavior of the intruder is given in Figure 4.44. The detailed data are shown in Appendix D. Though the result shows that the event IASendToS{eP$_S$(K$_{AS}$)} can occur either before or after the event

IRcvFromA{$eP_S(K_{AI})$}, different ordering of their occurrences will result in different situations: if the event IRcvFromA{$eP_S(K_{AI})$} occurs first, then this means that the intruder can actively start the event IASendToS{$eP_S(K_{AS})$}. On the other hand, if the intruder first performs IASendToS{$eP_S(K_{AS})$}, it has to wait for the occurrence of the event IRcvFromA{$eP_S(K_{AI})$}, which is not under its control. Thus it is quite possible that after long period of time waiting, the server *S* can then terminate the corresponding run of the SSL protocol. That is, the intruder may fail to attack the protocol by impersonating *A* to *S*.

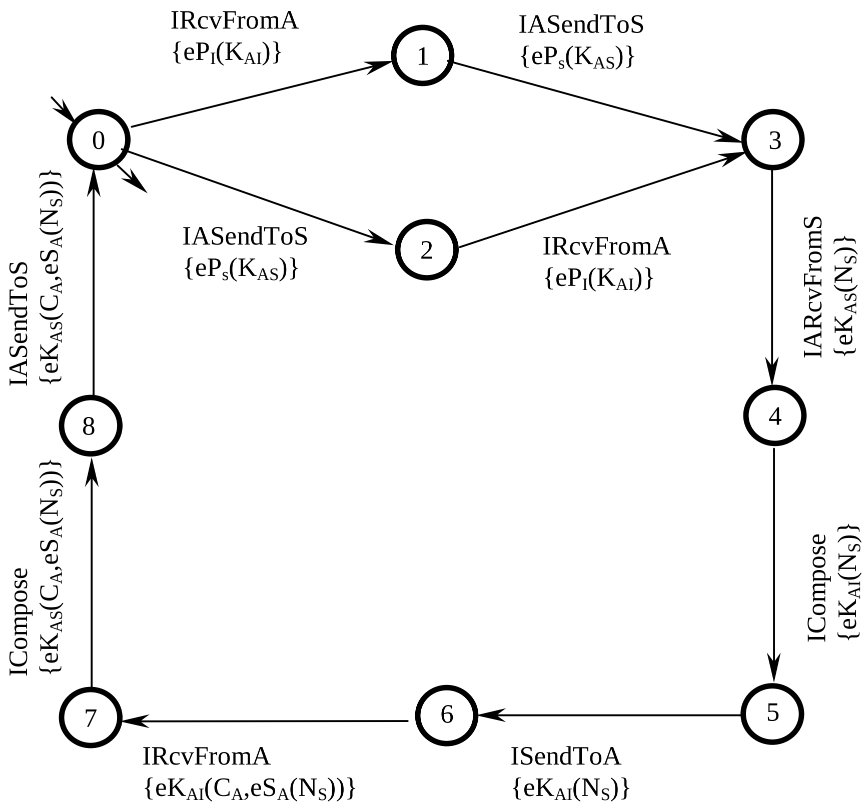

Figure 4.44 The intruder model for an attack on the SSL protocol

5 Automatic Scheme and Perspectives

In this chapter, we discuss the limiting factors for application of the proposed method described in Chapter 3. We then touch on the issue of constraints of a state-transition based modeling approach for the analysis of cryptographic protocols. Finally, we propose an automatic system for supporting the application of the proposed method.

5.1 Methods and Beyond

For cryptographic protocols, there are two major classes of analysis techniques: state-transition based approaches and logical approaches. While there have been successes with both approaches, each has its limitations and research continues into improving them. The algebraic approach [47,61-65] is to model the system as a state-transition based machine, to identify bad states and show that they cannot be reached under the assumptions of the protocol. A difficulty of the approach is how to fully define the abilities of the attacking process and what exactly constitutes a bad protocol state. In addition, there is a need for a computer tool to search the state space. The logical approach [7,34,46,66] models the belief and/or knowledge of each protocol principal and uses global inference rules to try to achieve the desired final beliefs after the protocol is complete. Difficulties here include how to translate between the logical language and the usual manner of protocol description.

Since the supervisory control framework of DESs is based on automata and formal language theory, our proposed method for the analysis of cryptographic protocols is, in essence, a state-transition based technique. As long as an approach uses a state-transition mechanism, no matter what name is used one has to explore those traces that lead to an insecure scenario. Therefore, it is necessary that the models created for the analysis of cryptographic protocols include those bad traces. However, such a task cannot be easily accomplished, because a combinatorial explosion of the number of states can result from the synchronous actions of all principals, including an intruder under consideration. As described in the previous chapter, in order to analyze an attack, we have to define a legal language based on the specification of a given protocol. The legal language tells us the sequences of events that should occur so as to enable the intruder to achieve its objective. Accordingly we need to try to build a so-called capability model for every message that is supposed to be sent by the intruder. We also have to build a number of models for related participants and channels. Then the intruder problem can be solved by the proposed method based on the supervisory control framework of DESs, and a resultant DFSA model will be produced for the intruder. Our proposed method is characterized by its goal-guided back-tracking mode, i.e., starting with a legal language, through constructing capability models, to building models for all participants and corresponding channel models. The state-space created by our method is bounded by the state-space of the participants that are involved in an intruder attack, the channels, the DFSA for the legal language, and the DFSA for the capability model. In contrast, in many methods discussed in [44], a larger number of participants and product runs are considered, thus increasing the computational efforts necessary to find an attack.

Though our proposed method helps a lot in reducing the modeling state space, the construction of the requisite DFSA models such as the legal language, capability models and models for all participants and channels still requires human experience, especially for the capability models. Thus a scheme for automatically creating the related DFSA models is desirable. Fortunately, since the DFSA models constructed by our methodology are simple and possess a high degree of modularity, it seems that an automatic scheme based on the technology of an expert system or artificial intelligence could be developed. In conjunction with the TCT software package, the following automatic scheme will make the proposed method fully automatic, so that it will become more attractive and practical.

5.2 Basic Principles

Although the state-transition based method can describe a run of a cryptographic protocol clearly and easily, it is well known that the subtlety of cryptographic protocols makes their design and analysis difficult. What we should ask is how many runs of the protocol and how many participants should be considered for an attack? There is a prohibitively huge number of combinatorial state-transitions which could be derived from almost any cryptographic protocol, because a good number of participants might take part in a number of runs of protocol. Up to now, we cannot assume an infinite amount of memory and disk space; neither can we assume an infinite-speed CPU (Central Processing Unit).

Therefore, this requires a production mechanism to derive new state-transitions based on current states, ultimately to construct required capability models, or to assert that an attack does not exist. The efficiency and effect of a production mechanism depends mainly on the way of presenting the set of state-transitions, where different representations result in different data structures and access methods. Using production rules to represent the set of state-transitions can help. Here we will give a principle for characterizing a set of valid transitions to get rid of storing a huge number of specific transitions. The Transition Production Principle is given as follows.

Principle 5.1 [Transition Production Principle]

If a run of a cryptographic protocol gives a set of transitions, involving a group of participants, then the following hold.

- Any role exchanges among members in this group of participants will produce a new set of transitions.
- Any member changes of the participants, where some new principals substitute for old principals, will produce a new set of transitions.

As for role exchange, take the Netscape's original SSL protocol for an example. Let Ω(A: Client; B: Server) denote the set of state-transitions produced in a run of the SSL protocol, where A is a client and B is a server. It is not difficult for us to obtain

$$\Omega(\text{A: Client; B: Server}) = \mathbf{\{}\text{ASendToB}\{eP_B(K_{AB})\}, \text{BRcvFromA}\{eP_B(K_{AB})\},$$

$\text{BSendToA}\{eK_{AB}(N_B)\}$, $\text{ARcvFromB}\{eK_{AB}(N_B)\}$,

$\text{ASendToB}\{eK_{AB}(C_A,eS_A(N_B))\}$, $\text{BRcvFromA}\{eK_{AB}(C_A,eS_A(N_B))\}$**}**.

Then we have

Ω(B: Client; A: Server) = **{**$\text{BSendToA}\{eP_A(K_{BA})\}$, $\text{ARcvFromB}\{eP_A(K_{BA})\}$,

$\text{ASendToB}\{eK_{BA}(N_A)\}$, $\text{BRcvFromA}\{eK_{BA}(N_A)\}$,

$\text{BSendToA}\{eK_{BA}(C_B,eS_B(N_A))\}$, $\text{ARcvFromB}\{eK_{BA}(C_B,eS_B(N_A))\}$**}**.

Any event of both Ω(A: Client; B: Server) and Ω(B: Client; A: Server) should be considered as a valid transition in the new system.

As for group member change, take the Needham-Schroeder authentication protocol for an example. Let Ω(A: Initiator; B: Responder; S: Server) denote the set of transitions produced in a run of the authentication protocol. Then we can get

Ω(A: Initiator; B: Responder; S: Server) = **{**ASendToS{A,B}, SRcvFromA{A,B},

$\text{SSendToA}\{eS_S(P_B,B)\}$, $\text{ARcvFromB}\{eS_S(P_B,B)\}$,

$\text{ASendToB}\{eP_B(N_A,A)\}$, $\text{BRcvFromA}\{eP_B(N_A,A)\}$,

BSendToS{B,A}, SRcvFromB{B,A},

$\text{SSendToB}\{eS_S(P_A,A)\}$, $\text{BRcvFromS}\{eS_S(P_A,A)\}$,

$\text{BSendToA}\{eP_A(N_A,N_B)\}$, $\text{ARcvFromB}\{eP_A(N_A,N_B)\}$,

$\text{ASendToB}\{eP_B(N_B)\}$, $\text{BRcvFromA}\{eP_B(N_B)\}$**}**.

By substituting an intruder *I* for the participant *A*, we could have

$$\begin{aligned}\Omega(\text{I: Initiator; B: Responder; S:Server}) = \mathbf{\{}&\text{ISendToS}\{I,B\},\ \text{SRcvFromA}\{I,B\},\\ &\text{SSendToI}\{eS_S(P_B,B)\},\ \text{IRcvFromB}\{eS_S(P_B,B)\},\\ &\text{ISendToB}\{eP_B(N_I,I)\},\ \text{BRcvFromI}\{eP_B(N_I,I)\},\\ &\text{BSendToS}\{B,I\},\ \text{SRcvFromB}\{B,I\},\\ &\text{SSendToB}\{eS_S(P_I,I)\},\ \text{BRcvFromS}\{eS_S(P_I,I)\},\\ &\text{BSendToI}\{eP_I(N_I,N_B)\},\ \text{IRcvFromB}\{eP_I(N_I,N_B)\},\\ &\text{ISendToB}\{eP_B(N_B)\},\ \text{BRcvFromI}\{eP_B(N_B)\}\mathbf{\}}.\end{aligned}$$

Then any event of both Ω(A: Initiator; B: Responder; S:Server) and Ω(I: Initiator; B: Responder; S:Server) should be considered a valid transition in the new system.

In addition, a legal language will introduce some additional transitions, which cannot be derived from the above Transition Production Principle. The validity of those transitions has to be proven by determining whether or not there is a corresponding capability model. Because an intruder is assumed to be able to intercept any message transferred over the network, among those additional transitions there are some transitions representing that the intruder has received certain messages, which should be considered valid transitions. Then the remaining transitions, indicating that the intruder sends certain messages, require special consideration. For those transitions that require the intruder to send certain messages, there will be corresponding capability models. If we can **prove** that there is no way to build a related capability model, then we can claim that the attack does not exist; otherwise, the capability model describes how the intruder composes the message under consideration. We establish some principles regarding how to compose messages.

Principle 5.2 [Cryptographic Operation Principle]

The following operations should be considered legal operations in terms of cryptography:

- Message Decryption: if a key K and a piece of message eK{X}are known to an intruder, then by decryption, the intruder can determine the value of X.
- Message Extraction: if {X,Y} is known to an intruder, then by extraction, the intruder can determine the value of X and the value of Y.
- Message Concatenation: if both X and Y are known to an intruder, then by concatenation, the intruder can compose {X,Y}.
- Message Encryption: if a key and a piece of message X are known to an intruder, then by encryption, the intruder can compose eK(X).
- Message Signature: if a key and a piece of message X are known to an intruder, then by digital signature, the intruder can compose sK(X).

For instance, with respect to Netscape's original SSL protocol, if an intruder knows the key K_{AI} and gets the message $eK_{AI}(C_A,eS_A(N_S))$, then it can get $\{C_A,eS_A(N_S)\}$ by message decryption. Up to this point, if it knows the key K_{AS}, then it can compose the message $eK_{AS}(C_A,eS_A(N_S))$ by message encryption. Consequently, the transition ICompose{ $eK_{AS}(C_A,eS_A(N_S))$ } should be considered a valid transition. By application of the Cryptographic Operation Principle iteratively, we can produce a number of transitions of type ICompose{…}.

In order to prevent fruitless search efforts, we propose following our so-called Termination Decision Principle. First a relevant concept is introduced below.

Definition 5.1 [Secure Fidelity]

A message block is said to be of *secure fidelity*, if this block contains at least one secret data item, a protocol run indicator, and an originator's identity, and if all of these are encrypted with a secure key. By "a secure key", we mean that the intruder does not know its corresponding decrypt key. By "secret data item", we mean there is no way for the intruder to know it due to some secure measures, such as if the item travels over the network always in the form of cyphertext encrypted using a secure key. A protocol run indicator may be a nonce or unpredictable time stamp. Every run of a protocol has a different value of the indicator.

Principle 5.3 [Termination Decision Principle]

If a message is of secure fidelity, then there will be no capability model for this message. If such a message is required to be sent by an intruder according to a legal language, then an attack described by the legal language does not exist. Therefore the corresponding analysis process can be terminated with a conclusion that no such attack exists.

The soundness of Principle 5.3 can be reasoned as follows. Because the message block is of secure fidelity, it contains a secret data item, a protocol run indicator, an originator's identity, and they are bound a secure key. An intruder has no way to achieve its impersonating objective by simply passing it on to another participant. Indeed, an intruder has to compose a similar message block with a different identity in the corresponding place for an originator's identity. However, the fact that the message block contains a secret data item prevents the intruder from composing a similar message block in that the intruder has no idea what should be put in place of the secret data item. The fact that the message block consists of a protocol run indicator prevents the intruder from replaying such a message block. Therefore, the claim that there is no corresponding capability model for an intruder to get this type of message block follows. For example, if the Needham-Schroeder authentication protocol is changed as follows, the attack described in section 4.2 can be prevented:

(1) $A \rightarrow S$: $\{A,B\}$

(2) $S \rightarrow A$: $\{eS_S(P_B,B)\}$

(3) $A \rightarrow B$: $\{eP_B(N_A,A)\}$

(4) $B \rightarrow S$: $\{B,A\}$

(5) $S \rightarrow B$: $\{eS_S(P_A,A)\}$

(6) $B \rightarrow A$: $\{eP_A(N_A,N_B,B)\}$

(7) $A \rightarrow B$: $\{eP_B(N_B)\}$

Note that the modification made to the message block in step (6) makes it to be of secure fidelity, i.e., it contains a secret data item, a protocol run indicator, an originator's identity, and they are bound with a secure key P_A. In fact, the nonce N_B is a secret data item, the nonce N_A can be used as a protocol run indicator, and B is an identity of the originator. So we can conclude that the attack defined by the legal language described in section 4.2 does not exist. If the identity of a principal is essential to the meaning of a message, it is prudent to mention the principal's name explicitly in the message.

It is necessary for us to establish a reasoning mechanism to be able to build capability models based on the Transition Production Principle and the Cryptographic Operation Principle. In the next section, we will give a description of an automatic scheme for application of our proposed systematic method to analyses of cryptographic protocols.

5.3 Automatic Scheme

Based on the discussion in Section 5.2, we propose an automatic scheme for the application of our systematic method. The automatic system as a whole is sketched as Figure 5.1.

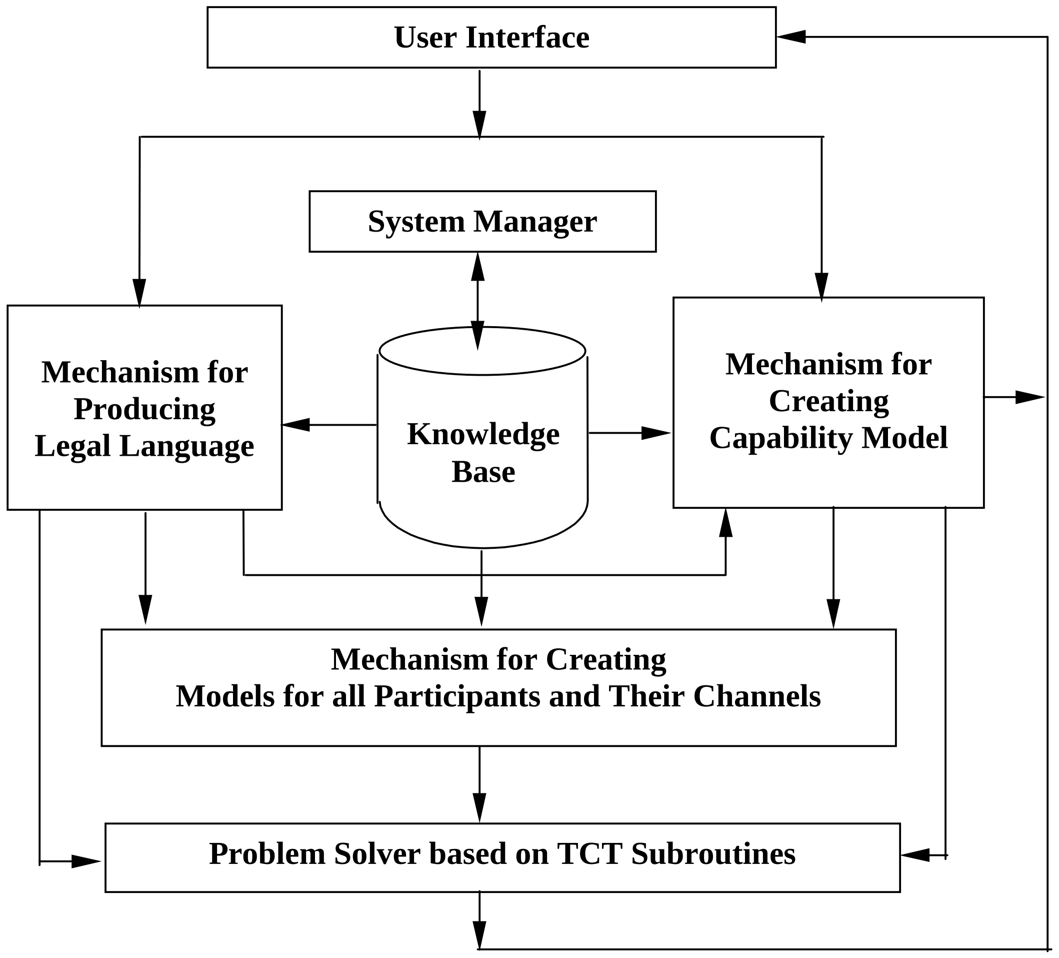
User Interface
System Manager
Mechanism for Producing Legal Language
Knowledge Base
Mechanism for Creating Capability Model
Mechanism for Creating Models for all Participants and Their Channels
Problem Solver based on TCT Subroutines

Figure 5.1 A system for supporting the application of our proposed method

As shown in Figure 5.1, the scheme consists of the following major components:

- An interface for a user to input a specification of a protocol and get an analysis result. The result will provide an answer about whether or not there are some attacks. If there are one or more attacks, the result should provide DFSA models, showing how the intruder achieves its objectives.

- An automatic mechanism for producing various kinds of legal languages, each of which represents some attack. There may be a number of different attacks. It is necessary for us to establish principles regarding how to create a legal language based on the knowledge of existing attacks.
- An automatic reasoning mechanism for searching and building capability models for those transitions in the legal language that require an intruder sending certain messages. Principles 5.1 and 5.2 can be applied to create and analyze possible ways for an intruder to get or compose messages.
- An automatic modeling mechanism for creating models for all participants and channels. After analyzing the relevant legal language and capability models, it defines what role every participant except the intruder plays, in turn leading to a composite DFSA model for the plant under consideration.
- A problem solver for the Intruder Problem (Problem 3.1) which is based on Theorem 3.1, and TCT subroutines. It is the Meet and Parallel Compositions that make it possible to build an overall capability model from a number of individual capability models, and that make it possible to build an overall channel model from a group of individual channel models. In turn, this makes the Divide-and-Conquer method applicable. As a result, the capability models can be created individually and iteratively for every message required to be sent by an intruder. Also the channel models can be built individually for every pair of participants. The Divide-and-Conquer method is an efficient and effective way for solving a large-scale problem.

The analysis of cryptographic protocols is a large-scale problem in terms of possible states and transitions, if a state-transition based approach is used.

- A knowledge base for storing and retrieving rules for guiding the creation of legal language, capability models, and models for all participants and their channels. Note that the DFSA model for the behavior of the intruder will be produced by the Problem Solver.
- A system manager for creating and updating the knowledge base, so that the system has the flexibility to integrate future work on the analysis of cryptographic protocols.

Iteratively applying the proposed method until no new attacks are discovered will help achieve the goal of securing a cryptographic protocol. If the system finds that there is no way for an intruder to compose or get certain required messages in the phase of creating capability models, it follows that the corresponding attack described by a legal language does not exist. Therefore, in Figure 5.1 there is an arrow line leaving from the mechanism for creating capability models to the user interface.

6 Discussion and Conclusions

In this chapter, first we give a brief discussion about cryptographic protocols and their analysis methods. Then we summarize and review the contributions of this thesis. Finally we present some possible future work related to this thesis.

6.1 Discussion

It is well known that the security of a system that uses encryption relies not only upon the soundness of the encryption algorithm employed, but also upon the security of the protocols by which information is shared. Authentication and key exchange protocols are typically the initial step in setting up a secure communications session; they are therefore critical in ensuring that security is maintained. In fact, an authentication protocol is at the heart of any system involving business transactions over a computer network. Without it, so-called e-business cannot function. However, cryptographic protocols have been shown to be prone to errors of every kind, especially where used in open networks for authentication and related purposes. Even when they have been developed carefully by experts and reviewed carefully by other experts, they are often found later to have flaws that may go undiscovered for a long time. One problem is the combinatorial complexity of the messages that an intruder could generate. A systematic approach is proposed for the analysis of attacks to cryptographic protocols, which casts the analysis of an attack to a cryptographic protocol as a supervisory control design problem for a discrete-event system. It is the operations meet and parallel compositions of two DFSAs that make this proposed approach attractive. This is because a complex system of multiple states and transitions can be decomposed into a number of sub-systems, each of which has fewer states and transitions, through the meet and the parallel compositions.

The main problem with a state-transition based method lies in the complexity of the combinatorial explosion of transitions resulting from multiple runs of a protocol with a number of participants. In essence, it is not efficient and effective for a protocol analyst to list all possible transitions and states, which could result in a huge and intractable state-transition space. If all possible events and states are derived by back-tracking according to the analysis objective, the legal language defining such an objective, and the specification of a given cryptographic protocol, then the problem can be solved efficiently and effectively. Our proposed method is based on goal-guided back-tracking. Though there is still need of experience to construct relevant models for the application of the proposed approach, the principles for building those models have been discussed in this thesis. Based on those principles, one could develop an automatic scheme to construct a legal language using various kinds of attack scenarios, then establish corresponding capability models, and finally build models for all participants and channels for the communications among those participants. This would make applications of the proposed approach easy and practical.

Protocol analysis is very important, in particular for cryptographic protocols in that they are used to protect significant data. However, cryptographic protocols are themselves subtle, which leads to the Golden Rule for designers of cryptographic systems: never underestimate the cryptanalyst.

6.2 Contributions

In summary, the thesis contains the following contributions:

- This thesis establishes an effective systematic method to analyze cryptographic protocols based on the Supervisory Control Framework of DESs. This method is a goal-oriented method with back-tracking characteristics.
- By exploiting meet and parallel compositions of DFSAs, this thesis makes feasible the Divide-and-Conquer strategy.
- The above two contributions set up a solid basis for an effective and efficient system of analysis of cryptographic protocols. This thesis proposes an automatic scheme for the analysis of cryptographic protocols, through discussion of the issues in analysis methods. In addition, this thesis presents prerequisite principles for the automatic scheme, i.e., the Principles 5.1, 5.2 and 5.3. The proposed scheme is not yet fully automatic, since it still requires human experience, especially for updating the knowledge base when a new type of attack is found.
- The construction of an attack on Netscape's original SSL protocol appears to be novel, since we did not find any material about how an intruder could perform its attack on the SSL protocol. Abadi and Needham in their paper [56] mentioned a flaw

with the SSL protocol but did not give a description of how an intruder could execute an attack.

- The attack on the Tatebayashi-Matsuzaki-Newman key distribution protocol appears to be novel, and cannot be prevented by the countermeasure proposed by M. Tatebayashi, N. Matsuzaki and D. B. Newman [54].
- This thesis re-builds the attack [5] on the Needham-Schroeder authentication protocol and the attack [49] on the Meyer-Matyas key distribution protocol, which shows the soundness of the proposed method.

6.3 Future Directions

As discussed in Chapter 5, still more work has to be done to realize an automatic system for the analysis of cryptographic protocols. The following are some major tasks that remain:

- The development of a mechanism for creating legal languages. First it is necessary to summarize various kinds of existing attacks on cryptographic protocols, then set up production rules for building a legal language.
- The development of a mechanism for creating capability models. The relevant basic principles have been proposed.

- The development of a mechanism for building models for participants and channels. Though some basic principles have been proposed, there is more work to do, especially for reasoning about what role each participant plays in a given protocol.
- To develop a program for the Problem Solver using TCT subroutines.
- To establish a knowledge base, a system manager module, and user interface module.

In addition, advances in web browsers with processing capabilities and programming languages allow web designers to embed real programs into an HTML document. Downloading and executing code from anywhere on the Internet may bring security problems along with it. A systematic and thorough analysis of security flaws in the browsers and their related technology is desirable.

References

[1] C. Glasheen, G. Byrne, J. Gantz, "The global market forecast for Internet usage and commerce," [Online document], (1998 July), Available URL: http://www.idc.com/

[2] R. M. Needham and M. D. Schroeder, "Using encryption for authentication in large networks of computer," Communications of the ACM, vol. 21, no. 12, Dec., pp. 993-999, 1978.

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

[4] R. K. Bauer, T. A. Berson, and R. J. Feiertag, "A key distribution protocol using event markers," ACM Transactions on Computer Systems, vol. 1, Aug., pp. 249-255, 1983.

[5] G. Lowe. "An attack on the Needham-Schroeder public-key authentication protocol," Information Processing Letters, vol. 26, pp. 131-133, 1995.

[6] CCITT, CCITT draft recommendation X.509: the directory-authentication framework, version 7, Glucester: CCITT, Nov., 1987.

[7] M. Burrows, M. Abadi, and R. Needham, "A logic of authentication," ACM Transactions on Computer Systems, vol. 8, no. 1, Feb., pp. 18-36, 1990.

[8] G. E. Simons, "How to (selectively) Broadcast a Secret," In Proceedings of the IEEE Symposium on Security and Privacy, 1985, pp. 108-113.

[9] C. A. Meadows, "Formal verification of cryptographic protocols: a survey," In Proceedings of Advances in Cryptology-ASIACRYPT'94, LNCS 917, Springer-Verlag, 1994, pp. 135-150.

[10] T. Hwang and Y.H. Chen, "On the security of SPLICE/AS – The authentication system in WIDE Internet," Information Processing Letters, vol. 53 pp. 97-101, 1995.

[11] T. Hwang, N.-Y. Lee, C.-M. Li, M.-Y. Ko, and Y.-H. Chen, "Two attacks on Neuman-Stubblebine authentication protocols," Information Processing Letters, vol. 53, pp. 103-107, 1995.

[12] J. Clark and J. Jacob, "On the security of recent protocols," Information Processing Letters, vol. 56, pp.151-155, 1995.

[13] D. Gollmann, "What do we mean by entity authentication," In Proceedings of the 1996 IEEE Symposium on Security and Privacy, IEEE Computer Society Presss, May 6-8, 1996, pp. 46-54.

[14] H. Gilbert, D. Gupta, A. Odlyzko, and J.-J. Quisquater, "Attacks on Shamir's 'RSA for paranoids'," Information Processing Letters, vol. 68, pp. 197-199, 1998.

[15] W.-G. Tzeng, C.-M. Hu, "Inter-protocol interleaving attacks on some authentication and key distribution protocols," Information Processing Letters, vol. 69, pp. 297-302, 1999.

[16] B. Schneier, Applied Cryptography, 2nd Edition, Toronto: John Wiley & Sons, Inc., 1996.

[17] A. Salomaa, Public-key Cryptography, 2nd Edition, Berlin, New York: Springer, 1996.

[18] C. E. Shannon. "Communication Theory of Secrecy Systems," Bell System Technical Journal, vol. 28, Oct., pp. 656-715, 1949.

[19] W. Diffie and M. Hellman, "New directions in cryptography," IEEE Transactions on Information Theory, vol. 22, no. 6, pp. 644-654, 1976.

[20] I. F. Blake, Elliptic Curves in Cryptography, New York: Cambridge University Press, 1999.

[21] R. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," Communications of the ACM, vol. 21, no. 2 Feb., pp. 120-126, 1978.

[22] B. Pfitzmann, Digital Signature Schemes, Berlin: Springer, 1996.

[23] W. Ford, M. Baum, Secure Electronic Commerce: Building the Infrastructure for Digital Signatures and Encryption, Toronto: Prentice Hall, 1997.

[24] U.S. Department of Commerce, "The Emerging Digital Economy II," [Online document], (1999 June), Available URL:
http://www.ecommerce.gov/ede/

[25] Asia Pacific Economic Cooperation Electronic Commerce Steering Group, "Paperless Trading Initiative," [Online document], (1999 June), Available URL:
http://www.ecommerce.gov/apec/docs/paperless.html

[26] P. J. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete event processes," SIAM Journal of Control and Optimization, vol. 25, no. 1, pp. 206-230, 1987.

[27] W. M. Wonham and P.J. Ramadge, "On the supremal controllable sublanguage of a given language," SIAM Journal of Control and Optimization, vol. 25, no. 3, pp. 637-659, 1987.

[28] K. Rudie and W. M. Wonham, "Supervisory control of communicating processes," in Protocol Specification, Testing and Verification, X, L. Logrippo, R. L. Probert, and H. Ural, Eds. North-Holland: Elsevier Sci. Pub., 1990, pp.243-257.

[29] K. Rudie and W. M. Wonham, "Think globally, act locally: decentralized supervisory control," IEEE Transactions on Automatic Control, vol. 37, no. 11, Nov., pp. 1143-1169, 1992.

[30] S. Lafortune and E. Wong, "Modelling and analysis of transaction execution in database systems," IEEE Transactions on Automatic Control, vol. 33, no. 5, May, pp. 439-447, 1988.

[31] S. Balemi, G. J. Hoffmann, P. Gyugyi, H. Wong-Toi and G. F. Franklin, "Supervisory control of a rapid multiprocessor," IEEE Transactions on Automatic Control, vol. 38, no. 7, July, pp.1040-1059, 1993.

[32] M. Lawford and W. M. Wonham, "Equivalence preserving transformations for timed transition models," IEEE Transactions on Automatic Control, vol. 40, no. 7, July, pp. 1167-1179, 1995.

[33] M. Abadi, M. Burrows, B. Lampson, and G. Plotkin, "A calculus for access control in distributed systems," ACM Transactions on Programming Languages and Systems, vol. 15, no. 4, September, pp. 706-734, 1993.

[34] W. Mao and C. Boyd, “Towards formal analysis of security protocols,” in Proceedings of the 6th Computer Security Foundations Workshop, 1993, pp.351-364.

[35] Paul C. Van Oorschot, “Extending cryptographic logics of belief to key agreement protocols,” in Proceedings of the 1st ACM Conference on Communications and Computer Security, Faixfax, Virginia, November 1993, pp. 312-316.

[36] J. Gray and J. McLean, “Using temporal logic to specify and verify cryptographic protocols,” in Proceedings of the 8th IEEE Computer Security Foundations Workshop, 1995, pp. 108-116.

[37] C. A. R. Hoare, Communicating Sequential Processes, New Jersey: Prentice-Hall, 1985.

[38] A. W. Roscoe, The Theory and Practice of Concurrency, New Jersey: Prentice-Hall, 1997.

[39] G. Lowe, “Breaking and fixing the Needham-Schroeder public-key protocol using FDR,” in Tools and Algorithms for the Construction and Analysis of Systems, LNCS 1055, 1996, pp. 147-166, Berlin/New York: Springer-Verlag.

[40] G. Lowe and B. Roscoe, “Using CSP to detect errors in the TMN protocol,” IEEE Transactions on Software Engineering, vol. 23, no. 10, pp. 659-669, 1997.

[41] G. Lowe, “Casper: A compiler for the analysis of security protocols,” Journal of Computer Security, vol. 6, pp. 53-84, 1998.

[42] S. Schneider, “Security properties and CSP,” in Proceedings of the IEEE Symposium on Security and Privacy, 1996, pp. 174-187.

[43] M. Abadi and A. D. Gordon, "A calculus for cryptographic protocols: the spi calculus," Information and Computer, vol. 148, no. 1, January, pp.1-70, 1999.

[44] R. A. Kemmerer, "Analyzing encryption protocols using formal verification techniques," IEEE Journal of Selected Areas Communication, Vol. 7, pp. 448-457, 1989.

[45] C. Meadows, "Applying formal methods to the analysis of a key management protocol," Journal of Computer Security, vol. 1, no. 1, pp. 5-36, 1992.

[46] J. K. Millen, "The interrogator model," in Proceedings of the IEEE Symposium on Security and Privacy, 1995, pp. 251-260.

[47] L. C. Paulson, "The inductive approach to verifying cryptographic protocols," Journal of Computer Security, vol. 6, pp. 85-128, 1998.

[48] B. B. Nieh and S. E. Tavares, "Modeling and analyzing cryptographic protocols using Petri Nets," In Proceedings of Advances in Cryptology-AUSCRYPT'92, LNCS 718, Springer-Verlag, 1992, pp. 275-295.

[49] E. Doyle, S. E. Tavares, and H. Meijer, "Analysis of cryptographic protocols using colored Petri Nets," in Proceedings of the 18th Biennial Symposium on Communications, Kingston, ON., May 1996, pp. 194-200.

[50] W. Wonham, "Notes on Discrete-Event Systems," [Online document], (1999 April), Available URL:
http://www.control.utoronto.ca/people/wonham/wonham.html

[51] G. Barrett and S. Lafortune, "Bisimulation, the supervisory control problem and strong model matching for finite state machines," The University of Michigan,

Department of Electrical Engineering and Computer Science, Ann Arbor, MI, Technical Report No. CGR-97-05, October 1997.

[52] M. Heymann, "Concurrency and discrete event control," IEEE Control Systems Magazine, vol. 10, no. 4, pp. 103-112, 1990.

[53] C. H. Meyer and S. M. Matyas, Cryptography: a new dimension in computer data security, New York: John Wiley & Sons, 1982.

[54] M. Tatebayashi, N. Matsuzaki and D. B. Newman, "Key distribution protocol for digital mobile communication systems," in Proceedings of Advances in Cryptology-CRYPTO'89, LNCS 435, Springer-Verlag, 1989, pp. 325-334.

[55] K. E. B. Hickman, "The SSL protocol," RFC, Netscape Communications Corp., version of Oct. 31, 1994.

[56] M. Abadi and R. Needham, "Prudent engineering practice for cryptographic protocols," IEEE Transactions on Software Engineering, vol. 22 no. 1, January, pp. 6-15, 1996.

[57] T. ElGamal, "A Public key cryptosystem and a signature Scheme Based on discrete logarithms," IEEE Transactions on Information Theory, vol. IT-31, July, pp. 469-472, 1985.

[58] W. Mao and C. Boyd, "On the use of encryption in cryptographic protocols," In Cryptography and Coding IV, P. G. Farrell (ed), pp. 251-262. IMA, 1995.

[59] International Organization for Standardization, Information technology – Security techniques – Data integrity mechanism using a cryptographic check function employing a block cipher algorithm, ISO/IEC 9797 Second Edition, April 1994.

[60] ISO/IEC International Standard 9594-8, Information technology – open systems interconnection – the directory, Part 8: Authentication framework, 1990

[61] R. Kemmerer, C. Meadows, and J. Millen, "Three Systems for Cryptographic Protocol Analysis," Journal of CRYPTOLOGY, vol. 7, pp. 79-130, 1994.

[62] S. Schneider, "Verifying authentication protocols with CSP," in Proceedings of 10th Computer Security Foundations Workshop, IEEE Computer Society Press, 1997, pp. 3-17.

[63] F. J. T. Fabrega, J. C. Herzog, "Strand space: Why is a security protocol correct?" in Proceedings of the 1998 IEEE Symposium on Security and Privacy, IEEE Computer Society Press, May 3-6, 1998, pp. 160-171.

[64] A. W. Roscoe, "Model-checking CSP," In A Classical Mind, Essays in Honour of C. A. R. Hoare, New Jersey: Prentice-Hall, 1994.

[65] F. Nielson, ed., ML with Concurrency, Berlin and New York: Springer, 1996.

[66] R. Kailar, "Reasoning about accountability in protocols for electric commerce", in Proceedings of the 1995 IEEE Symposium on Security and Privacy, IEEE Computer Society Press, May 8-10, 1995, pp. 236-250.

Appendix A TCT Data for the Analysis of the Meyer-Matyas Key Distribution Protocol

```
LEGAL_K    # states: 2    state set: 0 ... 1    initial state: 0

marker states:          0

vocal states: none

# transitions: 18

transitions:

[    0,  2,    1]  [    0,  4,    0]  [    0, 12,    0]  [    0, 14,    0]
[    0, 20,    0]  [    0, 21,    0]  [    0, 23,    0]  [    0, 24,    0]
[    0, 25,    0]  [    1,  4,    1]  [    1, 12,    1]  [    1, 14,    1]
[    1, 20,    1]  [    1, 21,    1]  [    1, 23,    1]  [    1, 24,    1]
[    1, 25,    1]  [    1, 27,    0]


CAPL_P    # states: 5    state set: 0 ... 4    initial state: 0

marker states:          0

vocal states: none

# transitions: 26

transitions:

[    0,  2,    0]  [    0,  4,    0]  [    0, 12,    0]  [    0, 14,    0]
[    0, 20,    1]  [    0, 24,    2]  [    1,  2,    1]  [    1,  4,    1]
[    1, 12,    1]  [    1, 14,    1]  [    1, 21,    3]  [    2,  2,    2]
[    2,  4,    2]  [    2, 12,    2]  [    2, 14,    2]  [    2, 25,    4]
[    3,  2,    3]  [    3,  4,    3]  [    3, 12,    3]  [    3, 14,    3]
[    3, 23,    0]  [    4,  2,    4]  [    4,  4,    4]  [    4, 12,    4]
[    4, 14,    4]  [    4, 27,    0]


INIT_A    # states: 2    state set: 0 ... 1    initial state: 0

marker states:          0

vocal states: none

# transitions: 2

transitions:

[    0,  2,    1]  [    1,  4,    0]
```

```
SERV_S    # states: 2    state set: 0 ... 1    initial state: 0

marker states:          0

vocal states: none

# transitions: 2

transitions:

[    0, 12,    1]  [    1, 14,    0]


CHNLALL    # states: 13    state set: 0 ... 12    initial state: 0

marker states:          0

vocal states: none

# transitions: 24

transitions:

[    0,  2,    1]  [    0, 14,    2]  [    0, 34,    3]  [    0, 38,    4]
[    0, 42,    5]  [    0, 46,    6]  [    0, 61,    7]  [    0, 63,    8]
[    0, 69,    9]  [    0, 71,   10]  [    0, 73,   11]  [    0, 75,   12]
[    1, 68,    0]  [    2, 66,    0]  [    3, 70,    0]  [    4, 74,    0]
[    5, 62,    0]  [    6, 64,    0]  [    7, 40,    0]  [    8, 44,    0]
[    9, 32,    0]  [   10, 12,    0]  [   11, 36,    0]  [   12,  4,    0]


INTRD    # states: 6    state set: 0 ... 5    initial state: 0

marker states:          0

vocal states: none

# transitions: 6

transitions:

[    0, 20,    1]  [    1, 21,    2]  [    2, 23,    3]  [    3, 24,    4]
[    4, 25,    5]  [    5, 27,    0]
```

Appendix B TCT Data for the analysis of Needham-Schroeder Authentication Protocol

LEGAL_K   # states: 3   state set: 0 ... 2   initial state: 0

marker states: 0

vocal states: none

# transitions: 72

transitions:

```
[    0,  2,    0]  [    0,  4,    0]  [    0,  6,    0]  [    0,  8,    0]
[    0, 10,    0]  [    0, 22,    0]  [    0, 24,    0]  [    0, 26,    0]
[    0, 30,    0]  [    0, 42,    0]  [    0, 44,    0]  [    0, 46,    0]
[    0, 48,    0]  [    0, 50,    0]  [    0, 52,    0]  [    0, 62,    0]
[    0, 63,    0]  [    0, 64,    0]  [    0, 65,    0]  [    0, 67,    1]
[    0, 68,    0]  [    0, 69,    0]  [    0, 72,    0]  [    0, 73,    0]
[    1,  2,    1]  [    1,  4,    1]  [    1,  6,    1]  [    1,  8,    1]
[    1, 10,    1]  [    1, 22,    1]  [    1, 24,    1]  [    1, 26,    1]
[    1, 28,    2]  [    1, 30,    1]  [    1, 42,    1]  [    1, 44,    1]
[    1, 46,    1]  [    1, 48,    1]  [    1, 50,    1]  [    1, 52,    1]
[    1, 62,    1]  [    1, 63,    1]  [    1, 64,    1]  [    1, 65,    1]
[    1, 68,    1]  [    1, 69,    1]  [    1, 72,    1]  [    1, 73,    1]
[    2,  2,    2]  [    2,  4,    2]  [    2,  6,    2]  [    2,  8,    2]
[    2, 10,    2]  [    2, 22,    2]  [    2, 24,    2]  [    2, 26,    2]
[    2, 30,    2]  [    2, 42,    2]  [    2, 44,    2]  [    2, 46,    2]
[    2, 48,    2]  [    2, 50,    2]  [    2, 52,    2]  [    2, 62,    2]
[    2, 63,    2]  [    2, 64,    2]  [    2, 65,    2]  [    2, 68,    2]
[    2, 69,    2]  [    2, 71,    0]  [    2, 72,    2]  [    2, 73,    2]
```

CAPL_P   # states: 8   state set: 0 ... 7   initial state: 0

marker states: 0

vocal states: none

# transitions: 138

transitions:

```
[    0,  2,    0]  [    0,  4,    0]  [    0,  6,    0]  [    0,  8,    0]
[    0, 10,    0]  [    0, 22,    0]  [    0, 24,    0]  [    0, 26,    0]
[    0, 28,    0]  [    0, 30,    0]  [    0, 42,    0]  [    0, 44,    0]
[    0, 46,    0]  [    0, 48,    0]  [    0, 50,    0]  [    0, 52,    0]
[    0, 62,    1]  [    0, 68,    2]  [    0, 72,    3]  [    1,  2,    1]
[    1,  4,    1]  [    1,  6,    1]  [    1,  8,    1]  [    1, 10,    1]
[    1, 22,    1]  [    1, 24,    1]  [    1, 26,    1]  [    1, 28,    1]
[    1, 30,    1]  [    1, 42,    1]  [    1, 44,    1]  [    1, 46,    1]
[    1, 48,    1]  [    1, 50,    1]  [    1, 52,    1]  [    1, 63,    4]
[    2,  2,    2]  [    2,  4,    2]  [    2,  6,    2]  [    2,  8,    2]
[    2, 10,    2]  [    2, 22,    2]  [    2, 24,    2]  [    2, 26,    2]
[    2, 28,    2]  [    2, 30,    2]  [    2, 42,    2]  [    2, 44,    2]
[    2, 46,    2]  [    2, 48,    2]  [    2, 50,    2]  [    2, 52,    2]
[    2, 69,    5]  [    3,  2,    3]  [    3,  4,    3]  [    3,  6,    3]
[    3,  8,    3]  [    3, 10,    3]  [    3, 22,    3]  [    3, 24,    3]
[    3, 26,    3]  [    3, 28,    3]  [    3, 30,    3]  [    3, 42,    3]
[    3, 44,    3]  [    3, 46,    3]  [    3, 48,    3]  [    3, 50,    3]
[    3, 52,    3]  [    3, 73,    0]  [    4,  2,    4]  [    4,  4,    4]
[    4,  6,    4]  [    4,  8,    4]  [    4, 10,    4]  [    4, 22,    4]
[    4, 24,    4]  [    4, 26,    4]  [    4, 28,    4]  [    4, 30,    4]
[    4, 42,    4]  [    4, 44,    4]  [    4, 46,    4]  [    4, 48,    4]
[    4, 50,    4]  [    4, 52,    4]  [    4, 64,    6]  [    5,  2,    5]
[    5,  4,    5]  [    5,  6,    5]  [    5,  8,    5]  [    5, 10,    5]
[    5, 22,    5]  [    5, 24,    5]  [    5, 26,    5]  [    5, 28,    5]
[    5, 30,    5]  [    5, 42,    5]  [    5, 44,    5]  [    5, 46,    5]
[    5, 48,    5]  [    5, 50,    5]  [    5, 52,    5]  [    5, 71,    0]
[    6,  2,    6]  [    6,  4,    6]  [    6,  6,    6]  [    6,  8,    6]
[    6, 10,    6]  [    6, 22,    6]  [    6, 24,    6]  [    6, 26,    6]
[    6, 28,    6]  [    6, 30,    6]  [    6, 42,    6]  [    6, 44,    6]
[    6, 46,    6]  [    6, 48,    6]  [    6, 50,    6]  [    6, 52,    6]
[    6, 65,    7]  [    7,  2,    7]  [    7,  4,    7]  [    7,  6,    7]
[    7,  8,    7]  [    7, 10,    7]  [    7, 22,    7]  [    7, 24,    7]
[    7, 26,    7]  [    7, 28,    7]  [    7, 30,    7]  [    7, 42,    7]
[    7, 44,    7]  [    7, 46,    7]  [    7, 48,    7]  [    7, 50,    7]
[    7, 52,    7]  [    7, 67,    0]


INIT_A    # states: 5    state set: 0 ... 4    initial state: 0

marker states:          0

vocal states: none

# transitions: 5

transitions:

[    0,  2,    1]  [    1,  4,    2]  [    2,  6,    3]  [    3,  8,    4]
[    4, 10,    0]


RESP_B    # states: 5    state set: 0 ... 4    initial state: 0

marker states:          0

vocal states: none

# transitions: 5

transitions:

[    0, 22,    1]  [    1, 24,    2]  [    2, 26,    3]  [    3, 28,    4]
[    4, 30,    0]
```

```
SERV_S    # states: 4    state set: 0 ... 3    initial state: 0

marker states:          0

vocal states: none

# transitions: 6

transitions:

[    0, 42,    1]  [    0, 46,    2]  [    0, 50,    3]  [    1, 44,    0]
[    2, 48,    0]  [    3, 52,    0]


CHNLALL    # states: 13    state set: 0 ... 12    initial state: 0

marker states:          0

vocal states: none

# transitions: 24

transitions:

[    0,  2,    1]  [    0,  6,    2]  [    0, 10,    3]  [    0, 24,    4]
[    0, 28,    5]  [    0, 44,    6]  [    0, 48,    7]  [    0, 52,    8]
[    0, 63,    9]  [    0, 67,   10]  [    0, 71,   11]  [    0, 73,   12]
[    1, 42,    0]  [    2, 62,    0]  [    3, 68,    0]  [    4, 46,    0]
[    5, 72,    0]  [    6,  4,    0]  [    7, 26,    0]  [    8, 64,    0]
[    9, 50,    0]  [   10, 22,    0]  [   11, 30,    0]  [   12,  8,    0]


INTRD    # states: 10    state set: 0 ... 9    initial state: 0

marker states:          0

vocal states: none

# transitions: 10

transitions:

[    0, 62,    1]  [    1, 63,    2]  [    2, 64,    3]  [    3, 65,    4]
[    4, 67,    5]  [    5, 72,    6]  [    6, 73,    7]  [    7, 68,    8]
[    8, 69,    9]  [    9, 71,    0]
```

Appendix C TCT Data for the Analysis of Tatebayashi-Matsuzaki-Newman Key Distribution Protocol

```
LEGAL_K    # states: 4    state set: 0 ... 3    initial state: 0

marker states:          0

vocal states: none

# transitions: 88

transitions:

[    0,  2,    0]  [    0,  4,    0]  [    0, 12,    0]  [    0, 14,    0]
[    0, 32,    0]  [    0, 34,    0]  [    0, 36,    0]  [    0, 38,    0]
[    0, 40,    0]  [    0, 42,    0]  [    0, 44,    0]  [    0, 46,    0]
[    0, 61,    1]  [    0, 66,    0]  [    0, 67,    0]  [    0, 68,    0]
[    0, 69,    0]  [    0, 70,    0]  [    0, 71,    0]  [    0, 73,    0]
[    0, 74,    0]  [    0, 75,    0]  [    1,  2,    1]  [    1,  4,    1]
[    1, 12,    1]  [    1, 14,    1]  [    1, 32,    1]  [    1, 34,    1]
[    1, 36,    1]  [    1, 38,    1]  [    1, 40,    1]  [    1, 42,    1]
[    1, 44,    1]  [    1, 46,    1]  [    1, 62,    2]  [    1, 66,    1]
[    1, 67,    1]  [    1, 68,    1]  [    1, 69,    1]  [    1, 70,    1]
[    1, 71,    1]  [    1, 73,    1]  [    1, 74,    1]  [    1, 75,    1]
[    2,  2,    2]  [    2,  4,    2]  [    2, 12,    2]  [    2, 14,    2]
[    2, 32,    2]  [    2, 34,    2]  [    2, 36,    2]  [    2, 38,    2]
[    2, 40,    2]  [    2, 42,    2]  [    2, 44,    2]  [    2, 46,    2]
[    2, 63,    3]  [    2, 66,    2]  [    2, 67,    2]  [    2, 68,    2]
[    2, 69,    2]  [    2, 70,    2]  [    2, 71,    2]  [    2, 73,    2]
[    2, 74,    2]  [    2, 75,    2]  [    3,  2,    3]  [    3,  4,    3]
[    3, 12,    3]  [    3, 14,    3]  [    3, 32,    3]  [    3, 34,    3]
[    3, 36,    3]  [    3, 38,    3]  [    3, 40,    3]  [    3, 42,    3]
[    3, 44,    3]  [    3, 46,    3]  [    3, 64,    0]  [    3, 66,    3]
[    3, 67,    3]  [    3, 68,    3]  [    3, 69,    3]  [    3, 70,    3]
[    3, 71,    3]  [    3, 73,    3]  [    3, 74,    3]  [    3, 75,    3]


CAPL_P    # states: 3    state set: 0 ... 2    initial state: 0

marker states:          0

vocal states: none

# transitions: 69

transitions:

[    0,  2,    0]  [    0,  4,    0]  [    0, 12,    0]  [    0, 14,    0]
[    0, 32,    0]  [    0, 34,    0]  [    0, 36,    0]  [    0, 38,    0]
[    0, 40,    0]  [    0, 42,    0]  [    0, 44,    0]  [    0, 46,    0]
[    0, 62,    0]  [    0, 63,    0]  [    0, 64,    0]  [    0, 66,    1]
[    0, 68,    0]  [    0, 69,    0]  [    0, 70,    0]  [    0, 71,    0]
[    0, 73,    0]  [    0, 74,    0]  [    0, 75,    0]  [    1,  2,    1]
[    1,  4,    1]  [    1, 12,    1]  [    1, 14,    1]  [    1, 32,    1]
[    1, 34,    1]  [    1, 36,    1]  [    1, 38,    1]  [    1, 40,    1]
[    1, 42,    1]  [    1, 44,    1]  [    1, 46,    1]  [    1, 62,    1]
[    1, 63,    1]  [    1, 64,    1]  [    1, 67,    2]  [    1, 68,    1]
[    1, 69,    1]  [    1, 70,    1]  [    1, 71,    1]  [    1, 73,    1]
[    1, 74,    1]  [    1, 75,    1]  [    2,  2,    2]  [    2,  4,    2]
[    2, 12,    2]  [    2, 14,    2]  [    2, 32,    2]  [    2, 34,    2]
[    2, 36,    2]  [    2, 38,    2]  [    2, 40,    2]  [    2, 42,    2]
[    2, 44,    2]  [    2, 46,    2]  [    2, 61,    0]  [    2, 62,    2]
[    2, 63,    2]  [    2, 64,    2]  [    2, 68,    2]  [    2, 69,    2]
[    2, 70,    2]  [    2, 71,    2]  [    2, 73,    2]  [    2, 74,    2]
[    2, 75,    2]


INIT_A    # states: 2    state set: 0 ... 1    initial state: 0

marker states:          0

vocal states: none

# transitions: 2

transitions:

[    0,  2,    1]  [    1,  4,    0]


RESP_B    # states: 2    state set: 0 ... 1    initial state: 0

marker states:          0

vocal states: none

# transitions: 2

transitions:

[    0, 12,    1]  [    1, 14,    0]


SERV_S    # states: 7    state set: 0 ... 6    initial state: 0

marker states:          0

vocal states: none

# transitions: 8

transitions:

[    0, 32,    1]  [    0, 40,    4]  [    1, 34,    2]  [    2, 36,    3]
[    3, 38,    0]  [    4, 42,    5]  [    5, 44,    6]  [    6, 46,    0]


CHNLALL    # states: 13    state set: 0 ... 12    initial state: 0
```

```
marker states:        0

vocal states: none

# transitions: 24

transitions:

[    0,  2,    1]  [    0, 14,    2]  [    0, 34,    3]  [    0, 38,    4]
[    0, 42,    5]  [    0, 46,    6]  [    0, 61,    7]  [    0, 63,    8]
[    0, 69,    9]  [    0, 71,   10]  [    0, 73,   11]  [    0, 75,   12]
[    1, 68,    0]  [    2, 66,    0]  [    3, 70,    0]  [    4, 74,    0]
[    5, 62,    0]  [    6, 64,    0]  [    7, 40,    0]  [    8, 44,    0]
[    9, 32,    0]  [   10, 12,    0]  [   11, 36,    0]  [   12,  4,    0]


INTRD    # states: 84    state set: 0 ... 83    initial state: 0

marker states:        0

vocal states: none

# transitions: 256

transitions:

[    0, 68,    1]  [    0, 69,    2]  [    0, 71,    3]  [    1, 69,    4]
[    1, 71,    5]  [    1, 75,    0]  [    2, 68,    4]  [    2, 70,    6]
[    2, 71,    7]  [    3, 66,    8]  [    3, 68,    5]  [    3, 69,    7]
[    4, 70,    9]  [    4, 71,   10]  [    4, 75,    2]  [    5, 66,   11]
[    5, 69,   10]  [    5, 75,    3]  [    6, 68,    9]  [    6, 71,   12]
[    6, 73,   13]  [    7, 66,   14]  [    7, 68,   10]  [    7, 70,   12]
[    8, 67,   15]  [    8, 68,   11]  [    8, 69,   14]  [    8, 71,   16]
[    9, 71,   17]  [    9, 73,   18]  [    9, 75,    6]  [   10, 66,   19]
[   10, 70,   17]  [   10, 75,    7]  [   11, 67,   20]  [   11, 69,   19]
[   11, 71,   21]  [   11, 75,    8]  [   12, 66,   22]  [   12, 68,   17]
[   12, 73,   23]  [   13, 68,   18]  [   13, 71,   23]  [   13, 74,    0]
[   14, 67,   24]  [   14, 68,   19]  [   14, 70,   22]  [   14, 71,   25]
[   15, 61,   26]  [   15, 68,   20]  [   15, 69,   24]  [   15, 71,   27]
[   16, 67,   27]  [   16, 68,   21]  [   16, 69,   25]  [   17, 66,   28]
[   17, 73,   29]  [   17, 75,   12]  [   18, 71,   29]  [   18, 74,    1]
[   18, 75,   13]  [   19, 67,   30]  [   19, 70,   28]  [   19, 71,   31]
[   19, 75,   14]  [   20, 61,   32]  [   20, 69,   30]  [   20, 71,   33]
[   20, 75,   15]  [   21, 67,   33]  [   21, 69,   31]  [   21, 75,   16]
[   22, 67,   34]  [   22, 68,   28]  [   22, 71,   35]  [   22, 73,   36]
[   23, 66,   36]  [   23, 68,   29]  [   23, 74,    3]  [   24, 68,   30]
[   24, 70,   34]  [   24, 71,   37]  [   25, 67,   37]  [   25, 68,   31]
[   25, 70,   35]  [   26, 62,   38]  [   26, 68,   32]  [   26, 71,   39]
[   27, 61,   39]  [   27, 68,   33]  [   27, 69,   37]  [   28, 67,   40]
[   28, 71,   41]  [   28, 73,   42]  [   28, 75,   22]  [   29, 66,   42]
[   29, 74,    5]  [   29, 75,   23]  [   30, 70,   40]  [   30, 71,   43]
[   30, 75,   24]  [   31, 67,   43]  [   31, 70,   41]  [   31, 75,   25]
[   32, 62,   44]  [   32, 71,   45]  [   32, 75,   26]  [   33, 61,   45]
[   33, 69,   43]  [   33, 75,   27]  [   34, 68,   40]  [   34, 71,   46]
[   34, 73,   47]  [   35, 67,   46]  [   35, 68,   41]  [   35, 73,   48]
[   36, 67,   47]  [   36, 68,   42]  [   36, 71,   48]  [   36, 74,    8]
[   37, 68,   43]  [   37, 70,   46]  [   38, 63,   49]  [   38, 68,   44]
[   38, 71,   50]  [   39, 62,   50]  [   39, 66,   51]  [   39, 68,   45]
[   40, 71,   52]  [   40, 73,   53]  [   40, 75,   34]  [   41, 67,   52]
[   41, 73,   54]  [   41, 75,   35]  [   42, 67,   53]  [   42, 71,   54]
[   42, 74,   11]  [   42, 75,   36]  [   43, 70,   52]  [   43, 75,   37]
[   44, 63,   55]  [   44, 71,   56]  [   44, 75,   38]  [   45, 62,   56]
[   45, 66,   57]  [   45, 75,   39]  [   46, 68,   52]  [   46, 73,   58]
[   47, 68,   53]  [   47, 71,   58]  [   47, 74,   15]  [   48, 67,   58]
[   48, 68,   54]  [   48, 74,   16]  [   49, 64,    0]  [   49, 68,   55]
[   49, 71,   59]  [   50, 63,   59]  [   50, 66,   60]  [   50, 68,   56]
[   51, 62,   60]  [   51, 67,   61]  [   51, 68,   57]  [   51, 71,   62]
[   52, 73,   63]  [   52, 75,   46]  [   53, 71,   63]  [   53, 74,   20]
[   53, 75,   47]  [   54, 67,   63]  [   54, 74,   21]  [   54, 75,   48]
[   55, 64,    1]  [   55, 71,   64]  [   55, 75,   49]  [   56, 63,   64]
[   56, 66,   65]  [   56, 75,   50]  [   57, 62,   65]  [   57, 67,   66]
[   57, 71,   67]  [   57, 75,   51]  [   58, 68,   63]  [   58, 74,   27]
[   59, 64,    3]  [   59, 66,   68]  [   59, 68,   64]  [   60, 63,   68]
[   60, 67,   69]  [   60, 68,   65]  [   60, 71,   70]  [   61, 62,   69]
[   61, 68,   66]  [   61, 71,   71]  [   62, 62,   70]  [   62, 67,   71]
[   62, 68,   67]  [   63, 74,   33]  [   63, 75,   58]  [   64, 64,    5]
[   64, 66,   72]  [   64, 75,   59]  [   65, 63,   72]  [   65, 67,   73]
[   65, 71,   74]  [   65, 75,   60]  [   66, 62,   73]  [   66, 71,   75]
[   66, 75,   61]  [   67, 62,   74]  [   67, 67,   75]  [   67, 75,   62]
[   68, 64,    8]  [   68, 67,   76]  [   68, 68,   72]  [   68, 71,   77]
[   69, 63,   76]  [   69, 68,   73]  [   69, 71,   78]  [   70, 63,   77]
[   70, 67,   78]  [   70, 68,   74]  [   71, 62,   78]  [   71, 68,   75]
[   72, 64,   11]  [   72, 67,   79]  [   72, 71,   80]  [   72, 75,   68]
[   73, 63,   79]  [   73, 71,   81]  [   73, 75,   69]  [   74, 63,   80]
[   74, 67,   81]  [   74, 75,   70]  [   75, 62,   81]  [   75, 75,   71]
[   76, 64,   15]  [   76, 68,   79]  [   76, 71,   82]  [   77, 64,   16]
[   77, 67,   82]  [   77, 68,   80]  [   78, 63,   82]  [   78, 68,   81]
[   79, 64,   20]  [   79, 71,   83]  [   79, 75,   76]  [   80, 64,   21]
[   80, 67,   83]  [   80, 75,   77]  [   81, 63,   83]  [   81, 75,   78]
[   82, 64,   27]  [   82, 68,   83]  [   83, 64,   33]  [   83, 75,   82]
```

Appendix D TCT Data for the Analysis of Netscape's Original SSL Protocol

LEGAL_K   # states: 3   state set: 0 ... 2   initial state: 0

marker states: 0

vocal states: none

# transitions: 36

transitions:

```
[    0,  2,    0]  [    0,  4,    0]  [    0,  6,    0]  [    0, 22,    0]
[    0, 26,    0]  [    0, 41,    1]  [    0, 42,    0]  [    0, 43,    0]
[    0, 46,    0]  [    0, 48,    0]  [    0, 49,    0]  [    0, 51,    0]
[    1,  2,    1]  [    1,  4,    1]  [    1,  6,    1]  [    1, 22,    1]
[    1, 24,    2]  [    1, 26,    1]  [    1, 42,    1]  [    1, 43,    1]
[    1, 46,    1]  [    1, 48,    1]  [    1, 49,    1]  [    1, 51,    1]
[    2,  2,    2]  [    2,  4,    2]  [    2,  6,    2]  [    2, 22,    2]
[    2, 26,    2]  [    2, 42,    2]  [    2, 43,    2]  [    2, 45,    0]
[    2, 46,    2]  [    2, 48,    2]  [    2, 49,    2]  [    2, 51,    2]
```

CAPL_P   # states: 6   state set: 0 ... 5   initial state: 0

marker states: 0

vocal states: none

# transitions: 49

transitions:

```
[    0,  2,    0]  [    0,  4,    0]  [    0,  6,    0]  [    0, 22,    0]
[    0, 24,    0]  [    0, 26,    0]  [    0, 41,    0]  [    0, 42,    1]
[    0, 46,    2]  [    1,  2,    1]  [    1,  4,    1]  [    1,  6,    1]
[    1, 22,    1]  [    1, 24,    1]  [    1, 26,    1]  [    1, 41,    1]
[    1, 43,    3]  [    2,  2,    2]  [    2,  4,    2]  [    2,  6,    2]
[    2, 22,    2]  [    2, 24,    2]  [    2, 26,    2]  [    2, 41,    2]
[    2, 48,    4]  [    3,  2,    3]  [    3,  4,    3]  [    3,  6,    3]
[    3, 22,    3]  [    3, 24,    3]  [    3, 26,    3]  [    3, 41,    3]
[    3, 45,    0]  [    4,  2,    4]  [    4,  4,    4]  [    4,  6,    4]
[    4, 22,    4]  [    4, 24,    4]  [    4, 26,    4]  [    4, 41,    4]
[    4, 49,    5]  [    5,  2,    5]  [    5,  4,    5]  [    5,  6,    5]
[    5, 22,    5]  [    5, 24,    5]  [    5, 26,    5]  [    5, 41,    5]
[    5, 51,    0]
```

INIT_A   # states: 3   state set: 0 ... 2   initial state: 0

marker states: 0

```
vocal states: none

# transitions: 3

transitions:

[    0,  2,    1]  [    1,  4,    2]  [    2,  6,    0]


SERV_S    # states: 3    state set: 0 ... 2    initial state: 0

marker states:           0

vocal states: none

# transitions: 3

transitions:

[    0, 22,    1]  [    1, 24,    2]  [    2, 26,    0]


CHNLALL    # states: 7    state set: 0 ... 6    initial state: 0

marker states:           0

vocal states: none

# transitions: 12

transitions:

[    0,  2,    1]  [    0,  6,    2]  [    0, 24,    3]  [    0, 41,    4]
[    0, 45,    5]  [    0, 51,    6]  [    1, 46,    0]  [    2, 42,    0]
[    3, 48,    0]  [    4, 22,    0]  [    5, 26,    0]  [    6,  4,    0]


INTRD    # states: 9    state set: 0 ... 8    initial state: 0

marker states:           0

vocal states: none

# transitions: 10

transitions:

[    0, 41,    1]  [    0, 46,    2]  [    1, 46,    3]  [    2, 41,    3]
[    3, 48,    4]  [    4, 49,    5]  [    5, 51,    6]  [    6, 42,    7]
[    7, 43,    8]  [    8, 45,    0]
```

VITA

Name: Shaowu Luo

Place and Year of birth: Hunan Province, China, 1963

Education:
1988: M. Eng. Department of Automation, Tsinghua University, Beijing, China

Work Experience:
1998-1999, Research Assistant, Department of Electrical and Computer Engineering, Queen's University, Kingston, Canada.
1997-1999, Teaching Assistant, Department of Electrical and Computer Engineering, Queen's University, Kingston, Canada.
1994-1997, Software Engineer, System Analyst, China National CIMS/ERC(Computer Integrated Manufacturing Systems/Engineering Research Center), Beijing, China.
1993-1994, Programmer, Research Assistant, Frankfurt University, Frankfurt/Main, Germany.
1988 – 1993, Programmer, System Analyst, China National CIMS/ERC, Beijing, China.

Awards:
1998-1999 Queen's University Graduate Award (QGA).
1997-1998 R.S. McLaughlin Fellowship(RSMCL).
1991-1992 Research Excellence Award in Hi-Tech of China National Science and Technology Commission.