DESIGN AND VALIDATION OF AUTHENTICATION SYSTEMS

Approved by:

Dr. V. S. Sukumaran Nair

Dr. Jacob Abraham

Dr. James G. Dunham

Dr. Richard V. Helgason

Dr. Jeff Tian

DESIGN AND VALIDATION OF AUTHENTICATION SYSTEMS

A Dissertation Presented to the Graduate Faculty of the

School of Engineering and Applied Science

Southern Methodist University

in

Partial Fulfillment of the Requirements

for the degree of

Doctor of Philosophy

with a

Major in Computer Engineering

by

Krishnan G. Indiradevi

(B.Tech., University of Kerala, 1988)
(M.S., Southern Methodist University, 1996)

August 04, 2000

ACKNOWLEDGMENTS

I wish to express my sincere gratitude and appreciation to my advisor Dr. Suku Nair for arousing my interest and curiosity in network security, for guiding me through the trying path of research, and for the numerous insightful and stimulating discussions. I am also very thankful to the other committee members, particularly Dr. Jacob Abraham, for the valuable suggestions and comments on my research work.

I also take this opportunity to thank my dear friend Dr. Sanda Harabagiu for her invaluable support and encouragement. My sincere thanks also to my friends Dr. Krish Pillai, Mr. Ravi Gupta, Mr. Hyun Cheul Kim, Dr. Hakki Candan Cankaya, and Dr. George Deprez.

The support and encouragement given by my superiors and colleagues at IP Mobile, Inc. (Richardson, TX), especially Mr. Balaji Holur and Mr. Alexander Garbuz, are also sincerely acknowledged.

Finally, I thank my loving parents for giving me a good education and instilling in me an interest in science and technology, and my beloved grand mother from whom I learned the value of patience and perseverance.

Without the care and support of all these people, this work would not have reached the form in which it is now. I am indebted to all of them.

Indiradevi, Krishnan G. B.Tech., University of Kerala, 1988
M.S., Southern Methodist University, 1996

Design and Validation of Authentication Systems

Advisor: Professor V. S. Sukumaran Nair

Doctor of Philosophy conferred August 04, 2000

Dissertation completed July 07, 2000

Protecting the confidentiality and integrity of communications has always been an important problem confronting individuals and organizations. *Authentication* is a crucial step in ensuring information security - in simple terms, authentication strives to provide proof that an entity is indeed who it claims to be, and that a given piece of information indeed originated from a claimed source. Along with mechanisms for access authorization and data confidentiality, authentication forms an important building block in virtually all security frameworks.

Authentication is performed through the execution of an *authentication protocol* comprising a predefined sequence of cryptographic handshake messages, and associated rules and criteria to verify claims of identity. A wide variety of authentication protocols have been proposed and used in the past. However, a disturbing number of these seemingly secure protocols have later been found to be flawed, in many cases, years after their wide use.

Attacks against authentication protocols often succeed even without any compromise occurring in the underlying cryptographic system. Instead, attackers resort

to manipulation of protocol messages, and replay of messages from old authenticated sessions. Such attacks tend to be very subtle, and are unlikely to be detected by simple procedures such as manual inspection of protocols. This challenge has triggered considerable interest in the research community to develop formal, rigorous techniques for the analysis and verification of these protocols.

This thesis presents the results of an investigation into potential vulnerabilities in authentication protocols, formulation of methods to detect such weaknesses, and development of techniques to defend against the exploitation of those weaknesses that might still escape careful design and thorough analysis.

To address the problem of modeling authentication protocols and detecting their weaknesses, two alternative approaches are developed. The first is a rigorous formal approach based on *finite state machine modeling*, but tends to be somewhat complex. The second approach is based on *event simulation*which is not as rigorous as formal verification, but is simpler, more flexible, and often gives close enough results. An important criterion in developing both these approaches has been to make them useful and usable to both researchers and practicing protocol designers alike.

Finally, *meta authentication* is proposed as a powerful framework to protect authentication protocols from attacks. With minimal overhead, the meta authentication scheme provides a safe execution environment for authentication protocols. Being highly scalable and distributed, it is attractive in large networks. Both an architecture and a protocol, to support a robust environment for the execution of authentication protocols are developed and the effectiveness of the scheme demonstrated.

The problems in network security continue to grow with the deployment of large complex networks and the development of sophisticated protocols. Problems are more subtle and harder to detect. It is evident that the need for reliable and efficient validation techniques is more relevant than ever. Network and information security remains a fairly open problem, and appears likely to remain so well into the future.

TABLE OF CONTENTS

LIST OF FIGURES

This thesis is dedicated to the exceptional men and women, of past and present, who have dedicated their lives into creating a secure information society for us all.

Chapter 1

INTRODUCTION

Information technology is the art and science of the generation, processing, storage, distribution, and use of information. It has proved vital to modern society, and continues to shape the existence and operation of social institutions and our day to day lives in unimaginable ways. Computer networks, being a very efficient means of information exchange, have emerged as the single most important resource depended on heavily by governments, corporations, and individuals to gather and disseminate increasing amounts of information. Regrettably, as more and more sensitive data is generated, stored and exchanged, there has also arisen the threat of unauthorized access and malicious manipulation of such information. Historically these illegitimate and destructive activities were a cause of worry only to governments and military establishments, unfortunately this is not true any more.

Recent years have seen even ordinary individuals and organizations become attractive targets for malicious activities, mostly owing to the increasing dependence on information systems and the proliferation of communication networks. The situation is more serious now than ever because of the potential availability of enormous computing resources to subversive entities as well who can launch very effective attacks

against security mechanisms. A bewildering variety of unlawful activities are being reported regularly. These range from fraudulent use of wireless phones and illegal access of personal information, to phony electronic fund transactions and subversion of automated control systems. This has raised the serious, and to a large extent still open, question of information and network security. There is wide agreement that systematic and accountable policies and techniques to enforce selective access are needed to have control over the generation, manipulation, and dissemination of data. Though there is not as much agreement regarding the exact nature of solutions to be employed, it is not very difficult to identify some basic requirements that a good security framework should meet. One such requirement is *authentication*, the establishment of identities.

Authentication essentially involves proving claims of identity or authenticity. In a general network security framework, it serves as the first point of contact. An entity that desires access to the network or the information therein, is first required to pass authentication. Similarly, a piece of information, before being accepted into the information pool, is subjected to authentication. This process is carried out through special *authentication protocols*. A weak authentication protocol can seriously jeopardize the security of an information system, by failing to prevent illegitimate access to its resources.

Over the years, many authentication protocols that were once believed to be fool proof have been shown to be vulnerable to a variety of attacks. This has led to considerable research into their formal analysis and design. However, despite their

relatively simple structure, these authentication protocols have proved to be very hard to formally validate. This, combined with the fact that many of the known formal validation techniques are very cumbersome and demand considerable expertise, have made the level acceptance of formal techniques by real world protocol designers somewhat disappointing.

In this thesis, we attempt to develop techniques for designing, modeling, and validating these protocols using paradigms that are more likely to be familiar to network and protocol designers.

The remainder of this chapter is devoted to clarifying relevant concepts and introducing the terminology used throughout the thesis.

1.1. Information Security

An information system is a collection of entities that generate, process, consume, and exchange data containing information. In its current interpretation, an information system is generally thought to consist of computing nodes and data repositories interconnected through a communications network. Thus, while considering the security of an information system, two broad aspects can be identified:

- *Computer security*, concerning the protection of computing resources against unauthorized use, and safeguarding of data from intentional damage, disclosure, or modification [2] [5] [42]. This is a research area in itself and is not dealt with further in this thesis.

- *Network (communication) security*, concerning the protection of data, and the information carried therein, during its transmission through computer networks and distributed systems [14] [16] [42]. Note that this also includes informational exchanges performed as part of ensuring computer security.

Security mechanisms are designed to control the ability of entities to access and manipulate data or other resources in the computing and communication environment. These entities, commonly referred to as *principals*, include *users* for whom, and on whose behalf, information processing is performed, *hosts* which are addressable nodes where the processing is done, and *processes* which are instances of software programs running on hosts and performing the actual processing [42]. Security mechanisms must be incorporated at each of these levels in order to ensure confidentiality and integrity of various services provided.

1.2. Compromise of Security

When a security mechanism fails, a *security compromise* is said to have occurred. Such failures may result from unintentional actions of *trusted entities* or from deliberate actions of *malicious entities*, often known as *intruders*. The subversion of individual hosts within the information network (*host compromise*), or communication channels within the network (*communication compromise*), may lead to security breaches. A security breach resulting from the planned actions of an adversary trying to subvert a system qualifies as an *attack*. A successful attack may lead to disclosure,

corruption, or destruction of data, misuse of system resources, and denial of services normally available to trusted entities. Communication networks are vulnerable to two general forms of attack:

- *Passive attacks* that may result in the breach of *confidentiality* of communication between legitimate entities. Through wire-tapping or traffic analysis, intruders try to extract useful information by observing the data in transit or by analyzing communication patterns.
- *Active attacks* targeting the *integrity* or *availability* of information being exchanged. In these more malicious forms of attack, apart from mere observation, intruders try to control information by modifying, deleting, inserting, or replaying data. Active attacks often force denial of service to legitimates entities.

To defend against the increasing threat of intrusion, all non-trivial information systems employ security services in one form or another.

1.3. Security Services

An information system should support a variety of services to ensure security at different levels [42] [48].

Authentication services perform authentication of communicating entities or sources of data. Authentication ensures a principal in a communication session, that another principal in the session, or the source of some received data, is what it is claimed to be. Often used as a precondition for further security services, authentication has a crucial role and is the main subject of the work presented in this thesis.

Access control services help safeguard system resources against unauthorized use. These services depend on authentication as a precondition. Based on security policies implemented in the environment, each authenticated principal (user, host, or program) is granted a set of access rights to different resources.

Data confidentiality services prevent unauthorized disclosure of information to observers or unintended recipients of data. *Connection confidentiality services*, or *selective field confidentiality services* may be used to protect an entire message carrying data, or parts thereof, respectively. Indirect gathering of information through traffic flow analysis may be prevented through *traffic flow confidentiality services* .

Data integrity services ensure that data received by an entity has not been tampered with during transit. *Connection integrity* guarantees the integrity of all data exchanged in a communication session, whereas *selected field connection integrity* limits such protection to only specific portions of transmitted data. It is to be noted that integrity does not imply confidentiality.

Non-repudiation services protect messages against possible denial of origination by senders, or denial of receipt by recipients. This is achieved through services that guarantee *proof of origin*, or *proof of delivery*.

The aforementioned security services are implemented using various *security mechanisms* in appropriate combinations.

1.4. Security Mechanisms

Common building blocks used in setting up security services include *encryption, digital signatures, notarization, access control mechanisms, traffic padding techniques*, and *routing control* [42].

Encryption (encipherment) involves using cryptographic techniques to encipher all or parts of exchanged data, to provide confidentiality and sometimes, to provide other security properties [48]. *Shared key cryptography* and *public key cryptography* which are the most important variants of cryptography form the basis of most security services. They are examined later in more detail.

Digital signatures are mechanisms by which a sender includes a non-forgeable *signature* in a transmitted message. Signatures enable recipients to verify the authenticity of data origin, and prevent senders from repudiating the message at a later point in time. Signature mechanisms are closely related to properties of cryptographic transforms, and will be discussed together later.

Notarization is a mechanism wherein a *trusted third party* gives testifiable assurance to communicating principals regarding certain properties of the data being exchanged. Assurances may be regarding message integrity, message origination time, or identities of origin and destination.

Access control mechanisms restrict and control access to various system resources by principals. Control is enforced using authenticated identities of principals and predefined *access control lists*. *Security audit trails* are often generated for

later analysis to detect suspicious activities including access right violation attempts. Based on access policies, different principals may be given varying levels of access to resources.

Traffic padding mechanisms aim to foil attempts to indirectly collect information through traffic flow analyses. They use spurious communication sessions, or data units to defeat statistical and other analysis techniques likely to be employed by eavesdroppers.

Routing control mechanisms allow dynamic rerouting of communication channels that may be under intrusion threat. They also include policies to allow only data meeting certain security criteria to pass through the network.

The focus of this thesis is on *authentication protocols*, protocols designed to establish identities of principals. Specifically, it addresses techniques to verify that authentication protocols achieve their design goals. Over the past several decades, a variety of authentication protocols have been proposed and used. However, many of them have been found to be flawed, often years after they were first published. Due to the crucial role authentication has in network security, there is considerable interest in research community in developing techniques for systematically analyzing and verifying protocols designed to perform authentication.

The main aim of this thesis is to develop a framework for formally modeling, analyzing, and validating authentication systems and protocols. Though various approaches have been proposed in the past (each with its own merits and weaknesses), a major concern has been that, due to the difficulty in applying these (often complex)

techniques, and the high level of expertise demanded, many real world protocol design initiatives tend to shy away from them. Therefore, one criterion in the work presented herein has been to follow paradigms which are likely to be more familiar to network and protocol designers. The other important aim of the thesis is to investigate and develop techniques for defending authentication protocols against various forms of attack.

The remainder of the thesis is organized as follows.

1.5. Outline

This chapter introduced the scope of network security, various network security services and mechanisms, the role of authentication in security frameworks, and potential challenges.

Chapter 2 investigates the process of *authentication* as applied to communication networks, including the goals and basic principles of operation. It also discusses several types of authentication protocols and examines potential attacks against them.

Chapter 3 introduces the application of formal methods for verifying the functionality of authentication protocols. Two important approaches, namely, *logic based* and *state based*, are discussed and their relative strengths and weaknesses identified. The chapter also serves as a brief survey of the application of formal methods in security protocols.

Chapter 4 discusses state machine approaches to modeling and analyzing protocols, *algorithmic state machines* (*ASM*) that offer an expressive medium for modeling authentication protocols, and a powerful verification technique (*property verification*) based on state machine principles.

Chapter 5 develops a framework (based on the concepts developed in Chapter 4) to verify authentication protocols, and illustrates its application through detailed modeling and analysis of a classic protocol - the *Needham-Schroeder Public Key* Authentication Protocol. The technique developed is demonstrated to be generic enough to be applicable to a wide variety of authentication protocols.

Chapter 6 investigates discrete event simulation as a potentially useful tool for validating authentication protocols, and illustrates its application on a classic protocol.

Chapter 7 introduces the concept of meta authentication, and develops a framework that provides a fail safe communication environment for executing general authentication protocols. The effectiveness of meta authentication is demonstrated by showing how it defends against various attacks.

Chapter 8 presents a road-map for future work and then concludes the thesis.

Chapter 2

AUTHENTICATION PROTOCOLS

Authentication forms the basis of most security services set up to provide *authenticity, confidentiality,* and *integrity* to information exchange among principals in a communications network. Authentication mechanisms function as 'gate keepers' to information systems.

Weak authentication can seriously endanger the security of entire networks. Therefore, it is invaluable to identify potential problems in authentication through investigations into relevant requirements, fundamental assumptions, and operational principles. To address this issue, this chapter provides a brief discussion on the principles, goals, implementation mechanisms, and potential vulnerabilities of general authentication systems and protocols.

2.1. Principles

In one form of another, most information systems and networks support authentication of *peer entities* and *data origin* [42] [48]. Authentication is necessary to prevent misuse of resources and services, to ensure data integrity, and to establish proof of origin of data.

Peer entity authentication enables an entity in a communication network to verify that a peer in a communication session is indeed the entity that it claims to be. Through this, it is assured that the peer is not trying to *masquerade* as another and that it is not trying to bypass authentication through a *replay* of earlier authenticated sessions. *Data origin authentication*on the other hand establishes the authenticity of the source of a message, assuring that the message really came from the claimed source.

Generally, authentication is a pre-requisite for further security mechanisms including:

- *Authorization*, the process of granting permissions to principals, including access rights for services and resources
- *Accountability*, to trace actions uniquely to involved principals.
- *Confidentiality services*, to prevent unauthorized disclosure of information or identities.
- *Integrity services*, to protect against unauthorized modification of messages during transit.
- *Non-repudiation services*, to provide *proof-of-origin* or *proof-of-delivery* of messages.

Three broad categories of authentication techniques are widely used in information systems, the specific choice depending on the application domain. They are:

- *Proof by knowledge*, where authentication is based on *something the claiming entity knows* (eg: pass words, identification numbers). By far, this is the simplest and most widely used technique in communication networks of present.
- *Proof by possession*, where authentication is based on *something the claiming entity possesses* (eg: keys, identity cards, other physical tokens). This technique is more expensive, but is used in environments where enhanced security is required.
- *Proof by property*, where authentication is based on some *property of the claimer* (eg: biometric properties such as voice patterns, fingerprints, and facial images). This is complex and very expensive, but may be required in situations demanding utmost security.

In this thesis, the scope of authentication is mostly limited to *proof by knowledge*, since it is the form most relevant in communication networks. The 'proof' in this case is established through *authentication protocols* that comprise special handshake messages between communicating principals. As these protocols are as vulnerable to eavesdropping as any other, cryptographic techniques are almost invariably used to protect this *knowledge verification phase* of communication sessions. Some cryptographic principles and terminology essential for further discussions on authentication mechanisms are discussed briefly in the following section.

2.2. Cryptographic Techniques

Cryptology, comprising *cryptography* and *cryptanalysis*, is the science of secure communications [48]. *Cryptology* is based on the principle that a sensitive *plain-text* message may be conveyed securely by transforming it using *cryptographic algorithms* into a *cipher-text* form, thereby hiding the carried information from unintended recipients. Only recipients possessing certain knowledge of the algorithm and transformation parameters will be able to recover the original message.

Cryptography involves the design of strong and efficient cryptographic algorithms resistant to eavesdropping; *cryptanalysis*, on the other hand, is the study into 'breaking' crypto-systems by analyzing and exploiting potential weaknesses in the algorithms.

Generally, the algorithms are publicly known and the cryptographic transformations between *plain-text* and *cipher-text* are determined by transformation parameters known as *keys* input into the algorithm. Thus, one fundamental assumption is that it is computationally infeasible for an adversary to deduce or guess the correct transformation parameters needed for recovering the plain-text message. Transformation of plain-text into cipher-text is performed through *encryption*, and the reverse transformation, through *decryption*. The transformation parameters (keys) are drawn from a potentially infinite set, and are usually denoted by K for encryption and K^{-1} for decryption. This basic principle is illustrated in Figure 2.1.

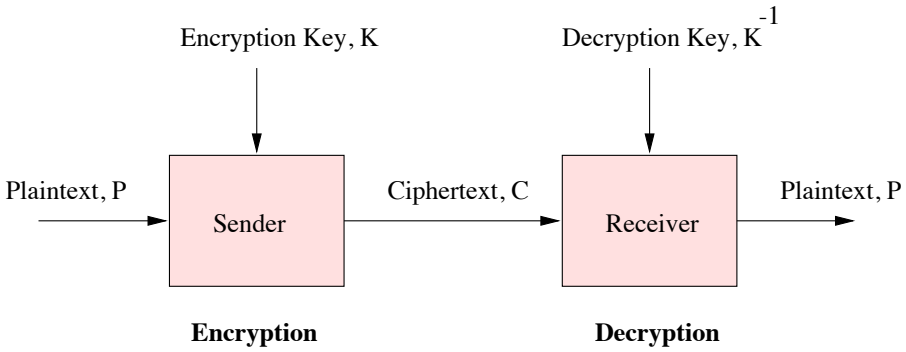


Figure 2.1. A basic cryptographic system

Authentication generally makes a *strong cryptography assumption*, meaning, it is impossible to decrypt an cipher-text message without the right key, and it is computationally infeasible for an adversary to deduce the right key. Following sections briefly discuss cryptographic techniques relevant in authentication systems and protocols.

2.2.1. Shared Key Cryptography

In *shared key (secret key) cryptography*, *shared (secret) keys* are established between communicating entities and used for message encryption and decryption. The security of the approach depends on the total secrecy of the keys. Its relevance to authentication is that, the use of the secret key is a demonstration of shared knowledge and hence identity. If the same key is used for both encryption and decryption, it is referred to as *symmetric key cryptography*. Though shared key systems can theoretically be operated solely by two principals through mutual consent and trust, in many cases the service of a *trusted third party (authentication server, AS)* is used to gener-

ate and distribute the keys. Though computationally efficient, the requirement that distinct shared keys be associated with every pair of communicating principals, makes this scheme hard to administer in large systems. Many widely used encryption standards such as *Data Encryption Standard (DES)*, *Triple DES*, and the *International Data Encryption Algorithm (IDEA)* are shared key systems.

2.2.2. Public Key Cryptography

Systems based on *public key cryptography* use *different* but mathematically related keys for encryption and decryption. Thus, they are *asymmetric key* systems. Each principal possesses a unique pair of keys, a *public key* which is made publicly known, and a *private key* known only to the principal.

Messages are sent securely by encrypting with the *public key* of the recipient. Only the intended recipient possessing the corresponding *private key* can decrypt and recover the original message. Conversely, in many of these schemes, data encrypted with a *private key* can be decrypted only with the corresponding *public key*. A message recovered in this manner could come only from the concerned principal who did the original encryption using the private key. This forms the basis of *digital signature* mechanisms. Its relevance to authentication is that, a principal may use the knowledge of its private key to prove its identity.

A trusted third party, known as *Certification Authority (CA)*, is generally used as a repository and distribution center of the public keys of all principals. Anyone

needing the public key of another may obtain it from the CA. The *CA* has a well known public key that enables it to send trusted signed replies to the requesting principals.

RSA (due to Rivest, Shamir, and Adleman) [44] is one of the most widely used public key cryptographic algorithms.

Public key systems are easier to administer as there is no requirement for every possible pair of principals to have distinct shared keys. However, the computational overhead of these systems is an order of magnitude higher than that of shared key systems.

Because of their comparative simplicity and efficiency, shared key systems are in wider use compared to public key systems in situations requiring only data confidentiality. However, their unique properties (eg: through digital signatures) make public key systems much more attractive in cases where authenticity is important.

In practice, most security systems use some combination of both public and shared key systems. In such systems, public key techniques are used initially to set up *session keys* which are subsequently used with shared key techniques to secure further communications. One good example is the emerging *IP Sec* (Internet Protocol Security) standard [3] [21].

2.2.3. One-Way Hash Algorithms

Hash functions are used to prove *integrity* by generating evidence that no modification has been done on a message during transit [12] [48]. Application of a *hash function* H on a message M produces a small *hash value* $H(M)$ of the message, called a *message digest*. Essentially a hash function maps a very large input space into a much smaller output space. Two important properties of a good hash function are its *one-way* nature and *collision resistance*. Thus, given M and $H(M)$, it is computationally infeasible to find a message M' such that $H(M') = H(M)$. In practice, hash functions are modified to accept a secret key as one of its inputs and generate *keyed hashes*. A message digest so produced is a convincing proof of integrity. The message digest is sent along with the original message, the recipient recalculates the hash value from the original message and compares it with the received value to verify that they match.

Cryptographic literature uses a variety of notations. The next section summarizes of the notations used in subsequent discussions and analyses in the thesis.

2.2.4. Notations

Trusted Principals (parties) in a protocol are denoted as A, B. A *server* (possibly a trusted third party), is indicated as S. A malicious user (or intruder) is represented as C, Z or I. $Z(A)$ denotes an attempt by intruder Z to impersonate trusted entity A.

Encryption/decryption key pairs are denoted by K, K^{-1}. In a *shared key system*, the key shared between principals A and B is denoted by K_{ab} (or simply as K when there is no confusion). In a *public key system*, the public and private keys of principal A are denoted by K_a and K_a^{-1}, respectively; an alternative notation is Ka_pub and Ka_prv.

Encryption of message M using key K is denoted in various ways as $E(K : M)$, $E(M) : K$, $(M)_K$, $\{M\}_K$, or $[M]_K$. Similarly, decryption of an encrypted message M' using K^{-1} is represented as $D(K^{-1} : M')$, $D(M') : K^{-1}$, $(M')_{K^{-1}}$, $\{M'\}_{K^{-1}}$, or $[M']_{K^{-1}}$.

Protocols use *nonces* or *time-stamps* to ensure freshness of messages. A *nonce* is a random number freshly generated by a principal and intended for use *only once* and only for the *current run* of the protocol. N_a and T_a indicate, respectively, a nonce and a time-stamp generated by principal A.

Sequences of messages in a protocols are denoted as $m1, m2, m3$, etc (or simply as $1, 2, 3$, etc., when there is no confusion).

Two ways of representing a message from A to B in a public key system are indicated below. The message includes the sender's name and an encrypted portion containing a nonce and some data.

$$m1 \quad A \longrightarrow B \quad : \quad A, \{N_a, data\}_{K_b}$$

$$OR$$

$$(1) \quad A \longrightarrow B \quad : \quad A, E(K_b : N_a, data)$$

Correspondingly, the recipient B, who possesses K_b^{-1}, decrypts the encrypted portion of the message and retrieves Na and *data* as indicated below:

$$\left\{ \{N_a, data\}_{K_b} \right\}_{K_b^{-1}} \quad \Rightarrow \quad Na, data$$

$$OR$$

$$D\left(K_b^{-1} : E(K_b : N_a, data) \right) \quad \Rightarrow \quad Na, data$$

2.3. Authentication Protocols

An *authentication protocol* is a sequence of messages designed to establish the authenticity of the principals involved in a communication session. The most common way to achieve authentication is by *proof of knowledge* based on secrets such as encryption keys. An authentication protocol guarantees that if the principals are really who they claim to be, then at the conclusion of the protocol, they will be convinced of the use of other principals' identities (due to the use of secrets) [10]. Also in many protocols, as a side effect, one or more shared secrets would have been established at the end of the authentication [10].

Despite some controversy existing in the literature over the precise meaning of authentication [18], it is generally agreed that one of the goals of an authentication protocol is to establish that [27]:

- Each principal believes in a secret shared with its peer (*first-level* belief).
- Each principal also believes in its peer's belief in the shared secret (*secondary-level* belief).

2.3.1. Types of Authentication Protocols

Authentication protocols have been classified in various ways [12]. One of the most widely used classifications is based on the cryptographic systems used:

- *Shared key authentication systems without trusted third party* , where the principals in the protocol initially share a secret key which is used for securing communication or for setting up further secrets. This necessitates each possible pair of communicating entities to share a secret key.
- *Shared key authentication systems with trusted third party* , wherein the need to share a secret with every possible recipient is obviated by employing a trusted third party called an *authentication server (AS).* The server shares secret keys with each principal in the system, and functions as an intermediary in authentication and key setup for any pair of principals in the system. The server often takes the responsibility for generating fresh *session keys* as part of authentication procedure and distributes them to the requesting principals, who then use it to secure subsequent communications.
- *Public key authentication systems*, based on the properties of *public key cryptography.* Public key encryption enables anyone knowing the *public key* of a principal to send secure messages to the principal. Only the intended principal possessing the corresponding *private key* will be able to recover the original messages. The public keys used must be current (as old keys could be possibly compromised), and genuine (fake keys could be distributed by some malicious

entity). Public key systems achieve this through a *Certification Authority (CA)* that is trusted to distribute public keys of all principals in the system, and also maintains a list of invalidated public keys. *CA* will have a well known public key to enable anyone to securely communicate with it.

Another popular classification is based on the 'direction' of authentication:

- *One-way (unilateral) authentication*, where only one party is assured of the authenticity of the other. No guarantee is available to the second entity regarding the identity of the first.
- *Two-way (mutual) authentication*, where the aim is to assure both parties of each other's identity through an interleaving of two one-way protocols or through more complex handshakes.

Some typical authentication protocols are described in the following section to illustrate their general operation. This discussion will be referred to later while analyzing their potential vulnerabilities.

2.3.2. Example Authentication Protocols

A wide variety of authentication protocols have been proposed, used, and studied over the past [12] [48]. The discussion in this section covers a few representative protocols since an understanding of their operation is essential in studying their potential vulnerabilities.

2.3.2.1. A Simple Challenge-Response Protocol

A simple 'friend or foe' protocol employing challenges and responses [12] to verify identity is shown in Figure 2.2.

$$\begin{array}{lllll} (m1) & A & \longrightarrow & B & : \quad N_a \\ (m2) & B & \longrightarrow & A & : \quad \{N_a\}_{K_{ab}} \end{array}$$

Figure 2.2. Simple challenge-response protocol

Two principals A and B sharing the key K_{ab} perform authentication without any trusted third party mediation. A (the *initiator*) sends to B, an initial *challenge* message containing a nonce N_a. B returns as the *response* to A, N_a encrypted with K_{ab} (presumably known only to A and B). A decrypts the response with K_{ab} and verifies that the nonce is the same as it sent. On the assumption that K_{ab} is known only to A and B, A concludes that it is indeed B at the other end.

This simple protocol provides only *one-way* authentication, it does not give B any assurance about A's identity. However, the protocol may be easily extended to provide *mutual* authentication.

2.3.2.2. Needham-Schroeder Shared Key Protocol

The *Needham-Schroeder shared key protocol* [39] is a widely studied *shared key protocol with a trusted third party.* It is the basis of many real-life protocols including the *Kerberos Authentication System* [41]. The protocol as shown in Figure 2.3 involves

two communicating principals, A and B, and a trusted third party, S (*authentication server, AS*).

$$
\begin{array}{lcccl}
(m1) & A & \longrightarrow & S & : \quad A, B, N_a \\
(m2) & S & \longrightarrow & A & : \quad \left\{ N_a, B, K_{ab}, \{K_{ab}, A\}_{K_{bs}} \right\}_{K_{as}} \\
(m3) & A & \longrightarrow & B & : \quad \{K_{ab}, A\}_{K_{bs}} \\
(m4) & B & \longrightarrow & A & : \quad \{N_b\}_{K_{ab}} \\
(m5) & A & \longrightarrow & B & : \quad \{N_b - 1\}_{K_{ab}}
\end{array}
$$

Figure 2.3. Needham-Schroeder shared key protocol

The trusted third party S shares the keys K_{as} and K_{bs} with principals A and B, respectively. A requests S for an authenticated session with B by sending nonce N_a. S then generates and returns a session key K_{ab} to A along with N_a, and an encrypted message (encrypted with K_{bs}) to be forwarded to B. The entire message is encrypted with K_{as} so that only A can read it. A extracts the new session key K_{ab} from the message and then forwards the portion encrypted with K_{bs} to B. On decrypting this part, B is also in possession of the session key K_{ab}, which is assured to be shared only with A (whose identity is also included in the received message). B then verifies the presence of A by sending a nonce N_b, upon which A responds with $(N_b - 1)$ as confirmation.

As shown later, this protocol is vulnerable to an attack that exploits a weakness in its freshness assumption.

2.3.2.3. Needham-Schroeder Public Key Protocol

The *Needham-Schroeder public key protocol* [39] is one of the most widely studied and analyzed authentication protocols. It is based on *public key cryptography* , and utilizes a trusted *certification authority (CA)* that supplies the valid public key of any principal to any requesting principal who wishes to use the key to perform authentication. Figure 2.4 shows the protocol messages.

$$
\begin{array}{lllll}
(m1) & A & \longrightarrow & S & : \quad A, B \\
(m2) & S & \longrightarrow & A & : \quad \{K_b, B\}_{K_s^{-1}} \\
(m3) & A & \longrightarrow & B & : \quad \{N_a, A\}_{K_b} \\
(m4) & B & \longrightarrow & S & : \quad B, A \\
(m5) & S & \longrightarrow & B & : \quad \{K_a, A\}_{K_s^{-1}} \\
(m6) & B & \longrightarrow & A & : \quad \{N_a, N_b\}_{K_a} \\
(m7) & A & \longrightarrow & B & : \quad \{N_b\}_{K_b}
\end{array}
$$

Figure 2.4. Needham-Schroeder public key protocol

In the protocol as shown, A wishes to have an authenticated session with B and contacts the certification authority S, which supplies A with the *public key* K_b of B (signed with its own *private key* K_s^{-1}). Assuming that the public key K_s of S is well known, A verifies that the public key of B indeed came from S. It is instructive to note that this encryption/decryption with the keys of S is not intended for *secrecy*, but is rather used as a *signature mechanism* to establish *data origin authentication*.

A then sends to B a nonce N_a encrypted with B's public key, meant to be read only by B who has the corresponding private key. B decrypts the message and recovers N_a. In order to communicate back to A, B then contacts S and obtains A's public key in the same way that A obtained B's.

To ensure *mutual authentication*, B also generates a nonce N_b and sends it along with N_a to A. This message encrypted under A's public key is decrypted by A, who verifies that N_a is the same as it had sent to B. If the verification succeeds, A assumes that it is indeed B at the other end of the channel. A then completes the protocol by sending back N_b to B, who also does a similar verification and concludes that it is indeed communicating with A. Thus mutual authentication is achieved. It may be noted that, as part of the process, two *secrets* (N_a and N_b) have been exchanged between A and B. These secrets may optionally be used by A and B to secure further communication between them.

For the purposes of analysis, it is convenient to assume that A and B have prior knowledge of each other's public key, thereby allowing the server S to be removed from the picture. Without loss of accuracy, this results in a simplified version the protocol as shown in Figure 2.5.

$$
\begin{array}{lcccl}
(m1) & A & \longrightarrow & B & : \quad \{N_a, A\}_{K_b} \\
(m2) & B & \longrightarrow & A & : \quad \{N_a, N_b\}_{K_a} \\
(m3) & A & \longrightarrow & B & : \quad \{N_b\}_{K_b}
\end{array}
$$

Figure 2.5. Needham-Schroeder public key protocol - simplified version

2.3.3. Threats to Authentication

In benign environments, most authentication protocols function properly. However, they may fail in the presence of adversaries (intruders) capable of observing and manipulating protocols messages. The situation is complicated because, quite often, a malicious entity may itself be a valid principal in the system. Under such circumstances, it is assumed that an intruder may have the ability to *observe, insert, modify,* or *delete* messages exchanged through the network. However, according to the *strong cryptography* assumption, the intruder *is considered incapable of extracting information* from an encrypted message, unless the corresponding key is in its possession through some means (eg: key compromise). Essentially, keys chosen through guess work are considered virtually useless because of this assumption.

In a malicious environment, one of the most important requirements is to ensure that a possibly intentional *replay* of old valid messages will not result in false authentication leading to the use of old and potentially compromised secrets. Hence, the *timeliness (freshness)* of information on which the protocol acts is very important. This raises the questions of *belief, trust, and delegation* [10]. When this condition is not met, an intruder may be able to subvert authentication even when there is no key compromise. A large number of known vulnerabilities in authentication protocols arise from weaknesses of this type.

The following section describes how intruders may be able to mount attacks against authentication mechanisms by exploiting potential weaknesses. This is very

important since it points to various scenarios to be considered in modeling and analyzing authentication protocols.

2.4. Attacks on Authentication Protocols

The process of authentication is often subtle and non-intuitive, involving such abstract concepts as *belief, trust,* and *delegation.* Because of this, it is often very hard to identify logical flaws and other weaknesses in authentication systems. As a result, an adversary may be able to attack and defeat authentication by exploiting vulnerabilities in protocol structure, message sequences, and timing relations. Considering also the possibility that an intruder may be a legitimate principal having unlimited access to the communication channel (thereby allowing *observation, interception, removal, modification*, and *insertion* of messages), it is difficult to clearly foresee the occurrence of an attack or its implications.

A successful attack on an authentication protocol could lead to several undesirable results. The intruder might manage to pass off as another principal by falsely convincing others of its assumed identity. Intruder could learn a current and valid shared secret (thought to be known only to the legitimate principals participating in the protocol). It is also possible that legitimate principals might be tricked into using old 'secrets' that are in the possession of the intruder (eg: an old secret that had been compromised).

2.4.1. Types of Attacks on Authentication Protocols

A large number of attacks that operate by exploiting various weaknesses in authentication protocols is known [7]. They include:

1. *Key guessing attack*

   In this brute force approach, an intruder tries to guess the right key from the entire key space. The attack may succeed when the choice of keys is not carefully made, or where the key space is small enough for an exhaustive search.

2. *Known plain text attack*

   Simple protocols in which parts of encrypted messages are publicly known (or can be easily deduced) are vulnerable to this attack. Using some known plain text (eg: names, addresses) and the corresponding ciphertext, an adversary may be able to break the cryptographic system and recover the key or other secrets.

3. *Chosen cipher-text attack*

   In this form of attack, an intruder manages to get legitimate principals to encrypt some carefully chosen data. The resulting cipher-text is then used to help break the cryptographic code.

4. *Replay attack*

   An intruder accumulates tables of messages between legitimate principals and replays them when the right opportunity arises. Essentially, an old message (from a past valid session) is passed off as current. Unsuspecting principals may accept such old message as fresh, potentially allowing the intruder to establish

falsely authenticated sessions. Principals may even be tricked into using old compromised keys, enabling the intruder to eavesdrop on all further 'encrypted' communication.

5. *Oracle session attack*

   An intruder uses a principal as an 'oracle' in its malicious actions against another. The intruder engages in authentication sessions with two principals and tricks one of them into performing the encryption and decryption required in the session with the second. Such attacks often succeed due to vulnerabilities in protocol message structure or message sequences.

6. *Parallel session attack*

   Intruder exploits a legitimate principal to act against itself. Intruder engages in two simultaneous authentication sessions with the same principal and manipulates one session to generate messages needed in the other. One of the sessions may eventually pass authentication criteria.

Of these, key guessing attacks, known plain-text attacks, and chosen plain-text attacks are actually attacks on the underlying cryptographic mechanism and not directly on the authentication protocol logic. Hence, based on the *strong cryptography assumption*, such attacks are generally not considered while analyzing authentication protocols, but are rather left to studies on cryptographic algorithms.

Attacks that exploit weaknesses in the *freshness* assumption (such as replay of old messages) are often collectively called *interleaving attacks* and constitute an important class of attacks on authentication protocols [7].

The following section describes some representative examples of attacks to expose the non-intuitive logic behind them, and thus serves to demonstrate the demand for developing effective techniques to verify authentication protocols and identify their weaknesses.

2.4.2. Example Attacks on Authentication Protocols

The following attacks on well known protocols illustrate how subtleties in the logics of authentication makes detection of flaws through simple methods difficult and quite improbable.

2.4.2.1. A Replay Attack

As discussed in Section 2.4.1, a *replay* (or *freshness*) attack occurs when an intruder replays messages from a previous valid session with the intension of tricking a principal in the original run of the protocol into entering a bogus authenticated session.

Section 2.3.2.2 discussed the *Needham-Schroeder shared key protocol* [39] that uses a trusted third party (authentication server) to establish a fresh session key between two parties. This protocol was found by Denning and Sacco [15] to be vulnerable to a replay attack. The attack exploits the weakness in the original protocol as shown in Figure 2.3, that the 3rd message ($m3$) delivering the new session key from the server does not carry any information regarding its *freshness (timeliness)*. This

enables an intruder to pass off an old instance of $m3$ (from a previous run of the protocol) as fresh. The danger is that the session key distributed via the old protocol could have been broken and might be known to the intruder [12].

$$\begin{array}{llcll} (m3) & Z(A) & \longrightarrow & B & : \{K'_{ab}, A\}_{K_{bs}} \\ (m4) & B & \longrightarrow & Z(A) & : \{N_b\}_{K'_{ab}} \\ (m5) & Z(A) & \longrightarrow & B & : \{N_b - 1\}_{K'_{ab}} \end{array}$$

Figure 2.6. Replay attack on Needham-Schroeder shared key protocol

Figure 2.6 illustrates the attack. Intruder Z, pretending as principal A, tries to establish a falsely authenticated session with B. Z replays an old message $m3$ from an earlier run of the protocol between A and B. It is conceivable that the old session key, K'_{ab}, might have been recovered by Z through cryptanalysis. The first two messages ($m1$ and $m2$) are not even generated because Z is not interested in involving the trusted server S. Since principal B has no way of knowing this, at the conclusion of the 5th message ($m5$), Z has succeeded in passing off as A. This is a failure of the authentication mechanism.

Denning and Sacco suggest *time-stamps* to ensure freshness of messages [15].

2.4.2.2. A Parallel Session Attack

By engaging in two or more concurrent sessions of a protocol, an intruder may be able to manipulate one session to generate messages required in another, thereby potentially leading to a false authentication [12].

The Needham-Schroeder public key protocol discussed in Section 2.3.2.3 was found by Lowe to be vulnerable to an attack of this form [12] [29]. The fact that the flaw was discovered only after 17 years since the publication of the original protocol, serves as enough justification for research into comprehensive and systematic verification techniques.

The attack, as illustrated in Figure 2.7, is based on the simplified version of the protocol shown in Figure 2.5. The malicious principal Z waits for an unsuspecting principal A to initiate an authentication session to it. Then, in the guise of A, Z starts a concurrent authentication session to another unsuspecting principal B. As the protocol flow shows, Z succeeds in passing off as A. Interestingly, the initial run of the protocol started by A to Z proceeds normally.

$$
\begin{array}{llcll}
(m1) & A & \longrightarrow & Z & : \{N_a, A\}_{K_z} \\
(m1') & Z(A) & \longrightarrow & B & : \{N_a, A\}_{K_b} \\
(m2') & B & \longrightarrow & Z(A) & : \{N_a, N_b\}_{K_a} \\
(m2) & Z & \longrightarrow & A & : \{N_a, N_b\}_{K_a} \\
(m3) & A & \longrightarrow & Z & : \{N_b\}_{K_z} \\
(m3') & Z(A) & \longrightarrow & B & : \{N_b\}_{K_b}
\end{array}
$$

Figure 2.7. Parallel session attack on Needham-Schroeder public key protocol

As the figure shows, two runs of the protocol are involved - one between A and Z (messages $m1, m2, m3$), and another between Z (posing as A) and B (messages $m1', m2', m3'$).

In $m1'$, Z simply forwards to B, the message it received from A, after re-encrypting with B's public key. B, seeing that the *initiator* field of the message is A, assumes it is talking to A and sends its response in $m2'$ (including the nonce N_b) encrypted under A's public key. Z, unable to decrypt this message, merely forwards it to A in $m2$. A, not knowing that the message is actually generated by B, accepts it and sends the final message $m3$ to Z with whom it is engaged in authentication. Z recovers N_b from this message and then sends it to B in $m3'$. B verifies N_b and is convinced that it is talking to 'A', where in fact it is Z at the other end. Z has thus successfully subverted authentication even *without any key compromise.*

It is clear from these examples that, simple verification based merely on inspection may not reveal subtle problems in authentication protocols. Evidently, more rigorous techniques are needed for verifying such protocols. The problem domain has several characteristics that make it suitable for the application of formal verification techniques. As one of the goals of this thesis is to develop techniques to validate authentication protocols, formal approaches that have been proposed and used in the past for the verification of these protocols are discussed in the next chapter.

In passing, it is also worth noting that many attacks may be prevented by prudent engineering practices as pointed out by Abadi and Needham [1], and through more cautious approaches as noted by Gong and Syverson [20].

Chapter 3

FORMAL VERIFICATION TECHNIQUES

As discussed in Chapter 2, the vulnerabilities encountered in authentication are often too complex and counter-intuitive for informal analysis. The severity of the problem is evident considering that weaknesses in several protocols were not uncovered until several years after the protocols were originally published. In the case of *Needham-Schroeder public key protocol*, it took 17 years to discover a serious flaw, despite the fact that the protocol had been very widely studied and analyzed. Evidently, more comprehensive and rigorous techniques are required for the analysis and verification of this class of protocols.

Formal verification is a powerful approach applicable to a wide variety of systems for the verification of correctness and functionality. Despite their power and rigor, formal techniques are not often considered very attractive (or even feasible) in many cases, because such techniques generally tend to be cumbersome, computationally very expensive, and demand considerable expertise.

There are certain characteristics of a problem domain to be examined in deciding whether formal verification would be effective. The following [32] are some helpful guidelines in determining if a problem domain is amenable to formal verification:

- the problem is not trivial
- the problem is not too large
- the problem search space is huge
- the problem is fairly well understood
- the problem is more or less clearly expressible
- flaws are too non-intuitive for simplistic approaches
- errors from flaws are severe enough to justify expensive analysis

By their very nature, cryptographic protocols, especially authentication protocols, are good candidates for formal verification. The flaws encountered are complex and counter intuitive, but are generally well contained enough for modeling and analysis to be tractable [32]. Through rigorous modeling and analysis, a protocol may be verified to provide reliable authentication under *realistic assumptions* (one such assumption being *strong cryptography* discussed in Section 2.2).

Over years, researchers have applied various techniques for formally analyzing and verifying cryptographic protocols. As indicated in Meadows [32] and Millen [35], most current approaches are either derived from or influenced by two classic formalisms:

- *Logic based approaches*, pioneered by Burrows, Abadi and Needham [11]. These are based on *specialized logics*, usually modal logics of beliefs in systems.
- *State based approaches*, pioneered by Dolev and Yao [17]. These use *state transition models* to express algebraic properties of systems.

The above two are briefly discussed in the following sections in order to understand their relative merits and weaknesses, and hence to help formulate an effective approach to verification and validation.

3.1. Logic Based Verification

Methods based on *logic* and *belief* form an important class of verification techniques that have been applied widely to authentication protocols. These use logics similar to those developed for the evolution of knowledge and belief in distributed systems [32].

Through repeated application of *inference rules* on carefully chosen *assumptions* about the system, and deduction of the *'meanings'* of protocol messages, logics can help verify security properties. As applied to authentication, one of the main properties of interest relates to beliefs in the 'goodness' of keys.

The major advantages of this approach are intuitiveness and the comparative ease with which it can be understood and applied. However, it has the serious disadvantage that subtle properties are likely to be missed in modeling, because of the high level of abstractions involved.

Application of logics to authentication protocols was pioneered by by Burrows, Abadi, and Needham who proposed a *'Logic of Authentication'* (known after them as the *BAN logic*) [10] that became the basis for considerable further research in the area.

3.1.1. BAN Logic

BAN logic is perhaps the best known and most influential among logic verification schemes for authentication protocols. The logic bases its inferences on basic assumptions regarding system properties and statements about messages involved in protocol runs. Application of BAN Logic involves:

- Assumptions about the *initial beliefs or knowledge* possessed by principals involved.
- Assumptions regarding the *knowledge embedded in the messages* of the protocol.
- *Inference rules* for deriving further beliefs or knowledge from existing beliefs, knowledge, and messages.

Analysis of a protocol starts with certain initial assumptions about the system (eg: beliefs about the secrecy of keys, trust in an authentication server's ability to provide fresh session keys, etc.). Subsequently, in a procedure known as *idealization*, each message in the protocol is rewritten in a form that makes explicit the beliefs or conveyed knowledge (eg: an acknowledgment may be transformed into a form that makes explicit the fact that the sender has seen the previous message). Based on the assumptions and the transformed messages, further beliefs evolving in the system can be derived by applying relevant inference rules (eg: 'knowledge about freshness of part of a message implies freshness of the entire message').

Following are some of the constructs used in BAN Logic to express beliefs and actions [11]:

- P *believes* X : principal P believes in the truth of the statement or assertion X and may act on it.
- P *sees* X : principal P has seen a message containing X (possibly after decryption).
- P *said* X : principal P once originated a message containing X.
- *fresh*(X) : the formula or expression X is fresh (eg: a nonce).
- $P \overset{K}{\longleftrightarrow} Q$: K may be used as a shared key between P and Q.
- $\overset{K}{\longmapsto} P$: K is a public key for P.
- $\{X\}_K$: X encrypted under key K.

Some of the inference rules (postulates) used in the logic are :

- *Message meaning rules*, used in interpreting messages to derive beliefs about their origin. For instance, in the case of *shared keys*, the postulate is:

$$\frac{P \textit{ believes } \mathrm{P} \overset{\mathrm{K}}{\longleftrightarrow} \mathrm{Q}, \quad \mathrm{P} \textit{ sees } \{\mathrm{X}\}_{\mathrm{K}}}{P \textit{ believes } \mathrm{Q} \textit{ said } \mathrm{X}}$$

meaning, if P believes K is a key it shares with Q and subsequently sees a message X encrypted under K, then P concludes that X must have been originated by Q at some point in the past.

- *Nonce verification rule*, stating that, if a message is fresh then its sender still believes in its truth. This is expressed as the postulate:

$$\frac{P \textit{ believes fresh}(\mathrm{X}), \quad \mathrm{P} \textit{ believes } \mathrm{Q} \textit{ said } \mathrm{X}}{P \textit{ believes } \mathrm{Q} \textit{ believes } \mathrm{X}}$$

meaning, if P believes X is a fresh (recent) message and that it was originated from Q, then P also believes that Q still believes in the truth of X.

Protocol idealization transforms actual messages into a form suitable for the application of the logic. This may be illustrated by the transformation:

$$A \longrightarrow B \ : \ \{A, K_{ab}\}_{K_{bs}}$$

$$\text{into:} \qquad A \longrightarrow B \ : \ \{A \overset{K_{ab}}{\longleftrightarrow} B\}_{K_{bs}}$$

indicating that the message conveys the fact that K_{ab} may be used as a shared key between A and B.

Through repeated application of inference rules on beliefs and message meanings, if some predefined *goal beliefs* (adequate to prove authenticity) can be derived, then the protocol is considered verified. Assuming that the ability to demonstrate the knowledge of a secret key K is sufficient for authentication, a minimal set of goal beliefs to be established for mutual authentication are:

$$A \quad \textit{believes} \quad A \overset{K}{\longleftrightarrow} B$$

$$\text{and,} \qquad B \quad \textit{believes} \quad A \overset{K}{\longleftrightarrow} B$$

Some protocols achieve only these beliefs, resulting in a weaker form of authentication. Others provide stronger authentication by establishing additional beliefs:

$$A \quad \textit{believes} \quad B \quad \textit{believes} \quad A \overset{K}{\longleftrightarrow} B$$

$$\text{and,} \qquad B \quad \textit{believes} \quad A \quad \textit{believes} \quad A \overset{K}{\longleftrightarrow} B$$

Failure to attain adequate beliefs points to potential security flaws that may be isolated by closer examination of the inference steps.

The BAN Logic has been used to verify correctness and to find previously unknown flaws in many authentication protocols. However, because of some inherent limitations imposed by its underlying assumptions, and due to the lack of sufficiently complex reasoning mechanisms, the BAN logic has not been very effective in handling certain classes of properties [32]. Yet, its relative simplicity has made it far more popular than many other comparatively more powerful logic systems.

Several modifications and variations to the approach followed by BAN logic have been proposed. Among these are, an extension *(GNY Logic)* by Gong, Needham and Yahalom [19], and the work by Syverson and Oorschot [49] to unify a number of logic based approaches *(SVO)*.

A slightly different approach to the problem is suggested by Kailar and Gligor [27], using *evolution of beliefs*.

3.1.2. Evolution of Beliefs

Despite its apparent similarity to BAN Logic, the technique based on *evolution of beliefs* described in [27] is more extensive and is able to handle additional cases including inter-domain authentication and establishment of beliefs in environments that lack sufficient jurisdiction.

It makes use of the concept of *knowledge sets*, sets that refer to principals who share the contents of a message or knowledge in a session. In a manner resembling a state machine, every pair of belief and action leads to another belief, essentially representing an evolution of beliefs. Like BAN logic, this method also employs inference rules that drive belief evolution. Using inference rules and evolving beliefs, the membership of knowledge sets is updated. Beliefs of principals about the shared secrets (*first level beliefs*) and their beliefs about the beliefs of other principals (*second level beliefs*) are dealt with explicitly and separately for added trust.

Through repeated application of the rules, if it can be shown that the involved principals indeed belong to knowledge sets adequate for authentication, the protocol is considered correct. On the other hand, if an intruder can be shown to eventually acquire membership in the same knowledge sets, then it points to a security flaw.

3.2. State Based Verification

This important category encompasses a large class of approaches, ranging from language theoretic methods with implicit state assumptions, to techniques that use explicit state representations (Petri nets, for instance). They study state changes in the properties of a system in terms of knowledge and data values. The system under investigation is modeled and viewed as an *algebraic system* operated by an intruder trying to find some valid *words* in the system (eg: keys).

A strong advantage of state based approaches is the ability to capture properties more realistically, a direct result of the fairly low degrees of abstraction used. There is however the disadvantage that, it is much harder to model and use as compared to logic based approaches and demands considerable expertise. Another problem is potential state explosion.

The foundation of most approaches in this class lies in the pioneering work by Dolev and Yao [17] and a model they proposed.

3.2.1. Dolev-Yao Model

Dolev and Yao proposed a mathematical model using language formalisms to describe and analyze cryptographic systems. The model assumes a network where intruders are capable of performing all legal operations allowed for normal users, and in addition may be able to read all traffic, alter and destroy messages, and create or insert new messages. The only limitation imposed is that any secret information (eg: keys of other users) is assumed to be initially unknown to intruders.

Protocol is modeled as a *language system* with a finite set of distinct *symbols* that may be used to compose arbitrary *words*. *Messages* are formed by *concatenation* of words. Using *reduction rules* (*term rewriting rules*) definable in the language, sequences of words may be manipulated so as to transform them into other valid sequences accepted in the language. These rules determining the behavior of the system could be built-in or specified by the protocol designer. An example of a built-

in rule would be: 'an encryption followed by the corresponding decryption cancel out'.

An intruder's goal is to find out *words of interest* in the system, such as session keys and encryption keys. Through manipulation of messages, the intruder tries to extract such words. Thus, the intruder may be viewed as manipulating a *term-rewriting system* [32], subjecting sequences to rewriting rules performed by legitimate principals. Proving security of the protocol is then essentially a *word problem* in a *term-rewriting system*, ie., determining whether a desired word could be produced by starting from known word sequences and applying reduction rules of the language.

Dolev and Yao [17] used these observations to develop restricted classes of secure public key protocols, and also algorithms to analyze them in terms of their properties as term-rewriting systems. In these protocols, messages of the following form are exchanged between principals [36]:

$$A \longrightarrow B \quad : \quad x_n \Big(\cdots x_2 \Big(x_1(data) \Big) \cdots \Big)$$

Here, x_i's are operators that encrypt, decrypt, append, or remove a 'name-stamp field'. The encrypted part of each message forms the *data part* for the next one, with the limitation that secure data may be introduced only in the first message. Protocols of this form are referred to as *name-stamp protocols*. A simple name-stamp protocol with only encryption and decryption operators is sometimes called a *cascade protocol*.

If an intruder is able to obtain the data part of the first message, the protocol is insecure. Dolev and Yao proved two conditions for a cascade protocol to be secure. The first condition is that the first message should contain an encryption. The second condition is that, any decryption if present in a message, should be matched with the corresponding encryption.

In addition, they provided a polynomial time algorithm for testing the security of name-stamp protocols.

Several systems have been developed that combine the theoretical basis of the Dolev-Yao model with the state information handling capabilities and descriptive power of more conventional protocol verification frameworks (eg: formal specification languages). The following are some of the most notable state based modeling approaches used in authentication protocol verification:

- *Reverse search*, wherein a compromised (insecure) state is assumed and a backward search is performed to determine if such a state could be reached from a valid initial state. Several research systems follow this paradigm, many of them using *Prolog* based inference and proof frameworks. The *Interrogator* [36] and the *NRL Protocol Analyzer* [31] follow this model.
- *CSP based formalism*, wherein the system is modeled using *Communicating Sequential Processes (CSP)* and then analyzed to prove security properties expressed as invariants [47]. A major flaw in the classic Needham-Schroeder public key protocol was uncovered using CSP based techniques [29].

- *State enumeration*, which is a brute force technique of searching all reachable states starting from an initial state. Though the search space may be theoretically infinite, the search itself can usually be confined to smaller spaces in most practical protocols [37].

The following sections briefly discuss a few verification systems that are representative of state based approaches.

3.2.2. NRL Protocol Analyzer

This tool developed at the Naval Research Laboratory by Meadows et al. [31] [32] is loosely based on the Dolev-Yao model. The tool models a protocol as an interaction between a set of state machines. An attempt is then made to locate security flaws by working backwards from a specified insecure state.

Unlike the Dolev-Yao model which does not support local states (eg: description of data values), the NRL analyzer extends the model to include local state variables of principals. This enables modeling of additional failures such as those occurring in a case where an intruder falsely convinces a principal that a word possesses certain properties (eg: intruder tricking a principal into using an old or compromised session key).

The tool is designed as an interactive *Prolog* program. Protocol is specified as a set of transitions of state machines. Each transition rule specifies the *words* to be input by the intruder, the required values of state variables, words output by the

principal as a result of transitions (and subsequently learned by the intruder), and the new values of state variables. Transition rules may also specify such events as the generation of new messages from old ones performed by the intruder through application of operations such as encryption and decryption. As in the Dolev-Yao model, words involved in transition rules follow a set of *reduction rules* that transform them into other words acceptable in the system.

To find a flaw, description of a compromised state is input to the tool in terms of the variables and words potentially known to the intruder. Using transition rules, subsets of words and variables are taken and an attempt is made to find all states that may *precede* the specified state. For each such state, the procedure may be repeated to find states further back in the sequence. During this back traversal, if an allowed initial state in the protocol is reached, the compromised state is indeed *reachable* during the operation of the protocol. This indicates the presence of a flaw.

The backward search may generate potentially an infinite number of states. Interactive Mechanisms to narrow down the search space are provided, including querying of only portions of the state to reduce the matching required. The results may then be generalized using the property that, if a portion of a state is not reachable, then the whole state is not reachable. The system also supports automatic search using heuristics to narrow down search space. Search is further streamlined by exploiting properties of formal languages.

3.2.3. Petri-net Based Models

High level Petri nets have been used to model authentication protocols and to perform reachability analysis for verifying security. This is appealing since Petri nets offer a convenient framework for representing and analyzing communicating entities.

Entities in the system including trusted principals, intruders, and channels, are modeled as communicating machines interacting through messages and represented in the form of a Petri net [4]. Models of the communicating entities are essentially functional descriptions of the protocol, with *places* holding state variables or messages, and *transitions* representing various actions based on place values. Entities source and sink messages during the protocol run, causing place values to change and transitions to fire.

Behavior of intruders could include a *learning mechanism*, capturing the fact that an intruder can keep as part of its state all the messages and message components it has seen. Using such acquired knowledge, the intruder might try to compose and send new messages or replay old messages, resulting in a series of *markings* (snapshots of the global state of the net).

A security breach is considered to have occurred under one of the following conditions :

- A value in a *secret place* of a legitimate principal's machine ends up in a *place* in the intruder's machine, meaning that the intruder has learned a secret.

- A value in a *place* in the intruder's machine ends up in a *secret place* in a legitimate principal's machine, meaning that the intruder has tricked the principal into accepting a possibly fake 'secret'.

Through reachability analysis, it can be determined if the system will reach a *marking* (global state view) that matches one of the aforementioned conditions. A match indicates the presence of a security flaw.

The use of *predicate/transition nets* (using a tool called *PROD*) to verify the security properties of Needham-Schroeder authentication protocol is outlined in [4]. *Colored Petri (CP) nets* have also been used in verifying authentication protocols.

3.2.4. State Enumeration Models

One obvious way to approach verification is to use brute force state enumeration wherein all states reachable from an initial state of the protocol are enumerated and checked for security properties. The major difficulty with this approach is potential state explosion, because an intruder's actions could theoretically depend on an infinite number of past protocol runs (through use of old messages or parts thereof). However, in general, many practical authentication protocols are small compared to general communication protocols. Hence, protocol flaws become evident even when considering only a limited number of protocol runs, though the number of states to be considered may still be quite large. Recent advances in verification techniques and better tools have made searching these large state spaces fairly manageable.

A recent work in this direction uses the $Mur\varphi$ (pronounced, 'Mur-phi') specification and verification system [37]. $Mur\varphi$ is a general purpose state enumeration tool used in the verification of several general protocols. The protocol to be verified is modeled in the $Mur\varphi$ language and the model is attributed with a specification of properties. Through explicit enumeration, the tool checks if all reachable states of the model satisfy given specifications. In the model, a start state is specified with initial values of global variables, and transitions are specified using rules that are boolean conditions with associated actions. A non-deterministic model is followed, since for example, in the case of authentication protocols, an intruder may have the choice of replaying any of a number of old messages. Desired properties of the modeled protocol are specified as invariants which must be true in every reachable state. Violation of an invariant points to a flaw in the protocol.

Verification through state enumeration can be made much more powerful and efficient by developing techniques that gracefully handle the state explosion problem and employ *implicit enumeration* rather than *explicit enumeration*. Also, in previous approaches, the designer needs to know precisely how to describe a compromised state. This needs considerable knowledge of the workings of the protocol, not only of the message sequences, but of the cryptographic aspects as well.

In the following chapter we present a technique that makes verification considerably more versatile and efficient. For this, we decouple the actual design and the properties of interest. The design is modeled using state machines as in other techniques. But unlike other approaches that express properties to be verified also

as part of the design state machine, our technique expresses the properties as separate state machines distinct from the design state machine. Such decoupling allows enhanced flexibility and efficiency in modeling and analysis. Flexibility results from the fact that it enables the use of one exhaustive design model which may be checked against various property machines, each expressing a different property of interest. This obviates the need for generating a new model every time a new property needs to be verified. Further, as will be shown, the decoupling allows the use of powerful *implicit state enumeration* techniques for state matching, resulting in a very efficient verification process.

Chapter 4

A VERIFICATION FRAMEWORK USING STATE MACHINES

Formal verification of a system such as an authentication system requires a 'medium' in which to model it with sufficient accuracy and flexibility, and analyze rigorously. Several formalisms have evolved over the past several decades and found extensive use in systems modeling and verification. These include *finite state machines*, *Petri nets*, *CSPs (communicating sequential processes)*, and *programming language models*.

In this chapter, we explore the application of *finite state machine* concepts in studying authentication protocols, and develop a framework for modeling and verifying their properties. After an overview of the terminology used in state machine techniques, their use in protocol modeling and verification is briefly discussed, noting the advantages and disadvantages. We then propose the *algorithmic state machine* concept as a powerful paradigm to model authentication protocols. A new technique is developed to perform verification using *property state machines*. The theory behind the technique and a detailed case study demonstrating its applicability are discussed in detail.

4.1. Terminology

Finite state machine (FSM) is a formalism of fundamental importance in the representation, design, and analysis of event driven systems. An FSM consists of a finite set of *states* and a set of *transitions* among the states [23]. Transitions occur based on a set of *inputs*; corresponding to each input, there exists a transition out of every state (possibly reentering the same state, implying no state change). An FSM has a distinguished *initial* state q_0 where it starts, and possibly one or more *final* (goal) states. In general, an FSM may have a set of *outputs* tied to its states or transitions. Associated with each FSM is a directed graph called the *transition diagram* whose vertices and arcs correspond to the states and transitions in the FSM.

A general FSM may be formally defined [23] as a 6-tuple $M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$, where Q is a finite set of *states*, Σ is a finite *input alphabet*, Δ is the *output alphabet*, δ is the *transition function* mapping $Q \times \Sigma$ to Q, λ is an *output function* mapping Q to Δ (when outputs are associated with *states*) or $Q \times \Sigma$ to Δ (when outputs are associated with *transitions*), and q_0 is the *initial state*. For each state q and input a, the *next state* is then given by $\delta(q, a)$. If outputs are associated with states (*Moore machines*), $\lambda(q)$ gives the output in state q. When outputs are associated with transitions (*Mealy machines*), $\lambda(q, a)$ gives the output during a transition from state q on input a.

The following section is a brief review of the application of FSM concepts in the modeling of communication protocols.

4.2. Modeling Communication Protocols Using FSMs

FSMs have been used widely to model and verify general communication protocols [9] [8] [13]. Modeling and analysis of complex computer network protocols using state machines and graph theoretic techniques was presented by Postel [43]. Most of these early attempts use *explicit state enumeration* to analyze protocols. A major problem encountered in such cases is *state explosion*. In contrast, in the technique adopted in our present work, state explosion is mostly avoided through *implicit state enumeration*. Following is a brief overview of the conventional approach used in modeling protocols with FSMs.

The idea is to model the system as consisting of a number of components interacting through messages. Each of the interacting components is a finite state machine, and comprises a number of states (describing the properties of the component) and transitions that move the component from one state to another. The system may then be considered as having a state-space that is a subset of the product-space of the individual component state-spaces. Often only a subset needs to be considered, since many of the combinations in the original product space may not be meaningful or valid in the specified protocol.

In communication protocols, state transitions are generally associated with message transmissions and receptions. Since a transmission from one component will be coupled with a corresponding reception in another, the whole system may be thought to work as a synchronous machine, though at a finer level of detail, the components

operate asynchronously with respect to each other. Thus, it is possible to apply state machine verification techniques developed for synchronous sequential machines to the modeling and verification of protocols as well.

The concept is illustrated below using the *alternating bit protocol* [6] [8], essentially a 2-phase protocol involving a *sender* and a *receiver* with state machines as shown in Figure 4.1.

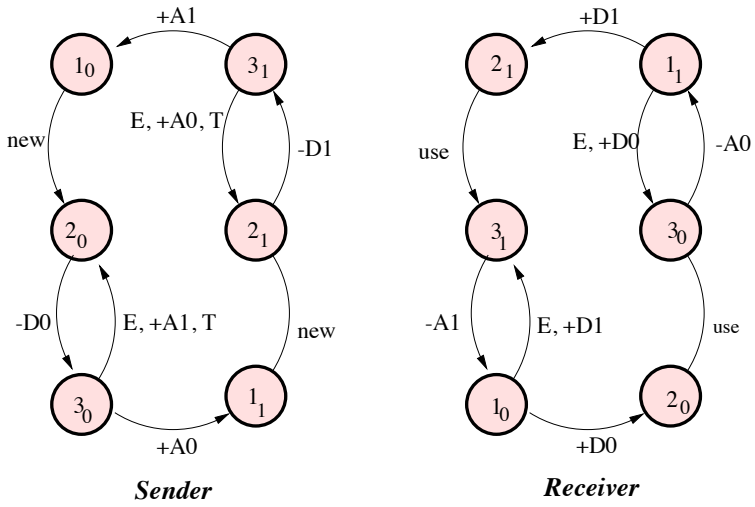


Figure 4.1. FSMs of 'sender' and 'receiver' in alternating bit protocol

Each data transfer cycle in the protocol occurs in 2 steps, a transmission followed by an acknowledgment ($D0, A0$ *or* $D1, A1$). Generation and consumption of messages are denoted as *new* and *use*. A transmission is indicated by '-' and a reception by '+'. E and T represent *error* (message corruption) and *timeout* (message loss), respectively. Considering a reliable medium with no errors or message loss, a *direct coupling* between *sender* and *receiver* can be defined by the transition pairs [8]:

$(-D0 \parallel +D0), (+A0 \parallel -A0), (-D1 \parallel +D1), (+A1 \parallel -A1)$

This means, each transmission from the sender has a corresponding reception in the receiver and vice versa. Thus, the two state machines can be assumed to undergo state transitions in lock step. Thus, the overall transition diagram of the system can be represented as in Figure 4.2. When the possibility of errors and losses is also accounted for, the number of states and transitions in the model increases considerably.

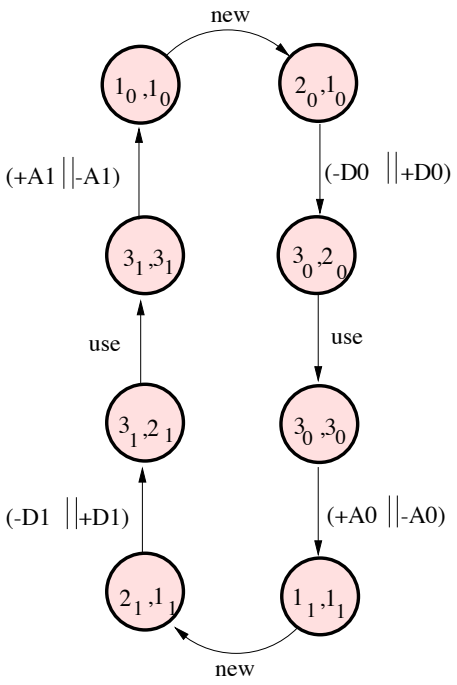


Figure 4.2. FSM of the overall system in alternating bit protocol

FSM representation of protocols provides a framework for studying various properties and for identifying errors in design or implementation. Design errors of particular interest in a general communication protocol are [50] :

- *State deadlock*, occurring when no transmission is possible from the present state of each process and no message is in transit, forcing each process to remain in its present state indefinitely.
- *Unspecified reception*, occurring when the design does not specify a possible reception. Such receptions are harmful as they may cause the system to transition to unknown states resulting in unpredictable behavior.
- *Non-executable interaction*, existing when the design specifies transmissions and receptions that cannot occur under normal conditions. Such interactions represent transitions that never occur, and are analogous to dead code in computer programs [50]).

Use of FSMs in modeling and analysis of communication protocols has been extensively studied [8] [13] [22] [50]. Despite being intuitive and clear, use of FSM representations is restricted by some of their limitations:

- *State space explosion*, meaning, the number of states and transitions tend to grow exponentially as the system grows. Consequently, the analysis becomes increasingly difficult, especially because most existing analysis techniques are based on explicit state enumeration and exhaustive state space search.
- *Limited expressive power* in comparison with other comparable formalisms. For instance, FSMs are considerably less expressive compared to general programming languages.

In general, these limitations restrict the use of FSMs to simple and small protocols. However, many of these limitations can be overcome to a great extent using modern techniques. This fact, combined with the inherent convenience and clarity of representation, and the amenability to well understood analysis techniques, makes the FSM formalism an attractive and powerful choice deserving serious consideration.

The remainder of the chapter discusses techniques to overcome the major limitations of FSM models, and develop an FSM based framework to model and verify authentication protocols. The discussion begins with the examination of a state machine extension known as *algorithmic state machine (ASM)* that offers remarkable improvement over the expressive power of conventional FSM models.

4.3. Algorithmic State Machines

Any system, such as a communication protocol or a data processor, that manipulates and processes data may be described by an *algorithm* which is a procedural sequence of steps producing a sequence of actions and outputs from a sequence of inputs. Accurate and clear representation of the algorithm is crucial in modeling and analyzing the system. A limitation of conventional FSM, as mentioned in the previous section, is its comparatively weak expressive power. *Algorithmic state machine (ASM)* is an extended model that attempts to overcome this limitation, and hence makes modeling of complex systems far more expressive and powerful [30] [40].

Most systems can be thought of as consisting of two distinct logical parts that interact to provide a specified functionality - a *data-part* and a *control-part*. The *data-part* provides common data processing functions including arithmetic, logic and other similar operations. In communication protocols, operations provided could also include encoding, decoding, etc. The *control-part* provides command signals to the data part to guide and sequence its various operations. In turn, the control-part takes feedback from the data-part and modifies its own control actions appropriately.

Algorithmic state machine (ASM) concept was developed specifically to combine the power of FSM (in expressing timing and state relations) and the clarity and expressiveness of flowcharts. An *ASM diagram* is fairly similar to a conventional flow chart, but with very different semantics and interpretation. Unlike a flowchart describing the procedural steps and decision paths of an algorithm, an ASM diagram codifies also the sequence of transitions among various states, events (*inputs*) that trigger transitions, and events generated (*outputs*) when the system is in a given state (*Moore machine* paradigm) or as it goes through a transition (*Mealy machine* paradigm). ASM diagrams are built from three types of elements [40]:

- *State box*, representing a state of the system and thus, equivalent to a node of a state diagram. For *Moore* type machines, it also specifies any output associated with the state. It has a single entry point and an exit point and is represented by a normal rectangle.

- *Decision box*, determining the transition to be taken out of a state for each relevant input in that state. Multiple input conditions are handled by cascaded decision boxes. It is represented as a diamond shaped box.
- *Conditional output box*, specifying outputs associated with a transition in a *Mealy* type machine. It is interposed between a decision box (which causes the transition) and a state box (into which the system moves), and is represented as a rounded box.

A state box and associated decision and control boxes together constitute an *ASM block*. An ASM block corresponds to a state in the equivalent state transition diagram and is represented as a dashed box.

Use of ASM is illustrated in Figure 4.3 showing the ASM representation of a simple system (a Mealy machine) and the equivalent state transition diagram. The system has only 2 states (A and B), 1 input (X), and 1 output (Z). The two dotted lines indicate ASM blocks corresponding to the two states in the transition diagram.

Obviously, the expressive power and clarity of ASM representation is considerably superior to that of simple FSMs. Later in the thesis, the ASM concept is used extensively to model the entities involved in authentication protocols.

The second major problem with conventional FSM approach is *state explosion*, making state space searches difficult and sometimes impractical. The following sections describe a powerful technique for addressing this problem.

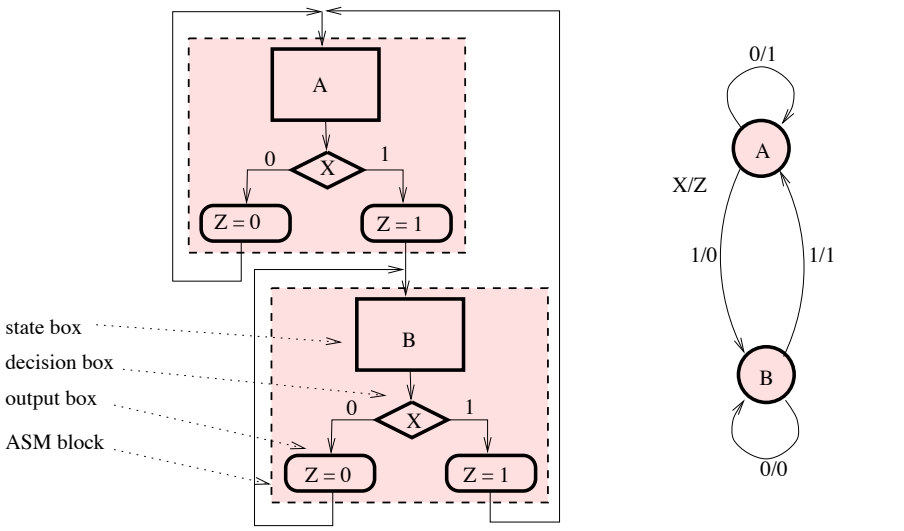


Figure 4.3. ASM and equivalent state diagram of a simple system

4.4. Design Verification

Verification may be applied at two stages in the development process of a system [24] [25]. The first is *design verification*, wherein an abstract high level design is verified for desired behavior or certain properties. The other is *implementation verification*, wherein a lower level implementation is verified for correct implementation of the corresponding high level design.

Since most serious flaws in authentication protocols are known to arise from design flaws rather than implementation problems, only design verification is addressed here. This section describes a technique to address problems in design verification, originally developed in the context of hardware verification [24].

The design is modeled as a (possibly incompletely specified) *deterministic* Mealy type finite state machine. However, unlike previous approaches that describe properties of interest in the design machine itself, this approach expresses the properties to be verified as separate state machines. These *specification machines (property machines)* are modeled as a type of *non-deterministic* state machine with some transitions having unspecified inputs. This partial specification of property machines allows very efficient implicit state matching. Using the design machine and the property machine, all states in the *design* that are *compatible* with the *start state* of the *specification* are computed. This step identifies states in the design having the same input-output behavior as that of the start state of the specification. Any matching design state, if found, indicates that the design exhibits the behavior expressed in the property machine. Conversely, absence of compatible states implies the absence of the specified behavior in the design. This completes the verification process.

As shown later in the chapter, this methodology provides an easy and intuitive specification format. However, its most important advantage is the power to *implicitly* search a given design's entire state space [26] [38].

In general, most approaches to design verification express properties to be examined as some form of *formulas in temporal logic* or as *automata*. Though the formulas in temporal logic can be quite expressive and powerful, a serious drawback in using them is that they can get extremely complicated and difficult to understand. This increased complexity may lead to misunderstanding of what property has actually been verified.

The technique discussed below was originally developed in the context of hardware verification [24] [25] and is intended to provide an easier and intuitive framework to verify various properties. We demonstrate later that this technique is very much applicable to protocol verification as well.

4.4.1. Verification Using Property State Machines

Generally, in *implementation verification*, a given design is checked to be *contained* in the implementation. Such containment proves that, for every state in the design, the implementation contains a corresponding state having the same *input-output behavior* as that of the design state (when the two machines start from these states). Such pairs of states are said to be *compatible*. Compatibility is slightly different from the notion of *equivalence* since compatibility allows the design to be partially specified, meaning, compatibility is checked only for input sequences that are *applicable* to the design.

On the other hand, the aim of *design verification* is to ensure that the design specification satisfies certain properties deemed necessary for the system under consideration. However, the same technique as used in implementation verification may be used here as well, using the design state machine and a property state machine codifying some property to be verified.

As discussed in Section 4.3, most design specifications can be seen to have a *data-path* and a *control-path*. The data-path models operations performed on data

(eg: arithmetic functions) whereas, the control-path models the sequencing and synchronization of various system operations. Control-path determines the actions of the data-path and, in turn, accepts inputs (status signals) from the data-path to modify its own sequence of actions. It may be noted here that the notion of *algorithmic state machines (ASM)* nicely maps to this model.

Generally, it is the data-path, depending on its *size* , that mostly contributes to potentially large state spaces. In many cases, it may be possible to avoid such state explosion by reducing the size of the data-path or even totally abstracting it out. However, while doing so, it is necessary to prove that this abstraction does not affect the validity of the design and that, properties such as boundary conditions which are verified for the scaled model also hold for the original design (this is perhaps more relevant for data processors than for protocols). This is achieved by ensuring the *functional correctness* of the data-path by modeling high level operations in the design using a library of boolean templates that have been proven correct with respect to first principles using theorem proving techniques. Thus, design verification does not need to deal with functional correctness. This allows a reduced model to be used, making the state space considerably smaller and manageable.

The reduced design state machine thus obtained can subsequently be used in conjunction with *property state machines* (specifying some properties to be verified) to prove that it possesses the required behavior.

A *property state machine* codifies a specified behavior to be verified as a state machine. Such a state machine consists of a sequence of inputs and corresponding

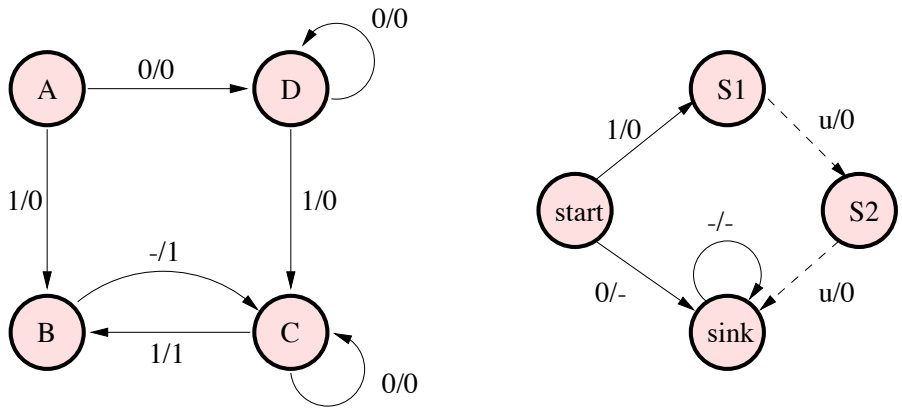
outputs, with a designated start state and possibly with *don't cares* in outputs or next states. A property machine differs from a conventional FSM by allowing some of its transitions to also have *unspecified* inputs, meaning, transition that are taken for *some* value of the inputs. This feature enables checking for transitions that satisfy the property, without prior knowledge of the inputs that cause them. To avoid ambiguity, only one such transition is allowed for a state, and it must be the only transition from that state. All transitions with undefined next states are absorbed by a *sink* state.

Two types of properties are important in verifying a design:

1. *Safety properties* of the form *"nothing bad will ever happen"*. A safety property is verified by checking for the existence of *bad behavior*. This implies checking *all* possible paths to be *good*, or in other words, checking if *any* path is *bad*.
2. *Liveness properties* of the form *"something good has to happen"*. A liveness property is verified by checking for states with no paths to *good* states. This implies checking if *any* path is *good*, or in other words, checking if *all* paths are *bad*.

The technique is illustrated using a simple system [25] with one input and one output as shown in Figure 4.4(a).

The goal is to verify that whenever the input goes high, the output will go high within two clock cycles. Verification may be done by checking for the occurrence of *bad* behavior. This implies examining whether there is any sequence of states where the output stays low for two clock cycles after the input goes high. The bad behavior is modeled by the *property machine (specification machine)* shown in Figure 4.4(b). As



(a) Design (b) Property specification

Figure 4.4. Example of state machine verification

the aim is to check for *any* such path, the two intermediate transitions in Figure 4.4(b) (shown in dashed lines) are left with unspecified inputs (u). Verification consists in finding if there is any state in the design state machine that is compatible with the *start* state of the specification state machine that represents the bad property. Design state D gives a match indicating that the bad sequence exists in the design, starting from D.

4.4.2. Compatible States and Property Verification

Design verification involves finding states in the design that are *compatible* with the *starting state* of the property specification, ie., states having the same input-output behavior.

The procedure [24] [25] uses the notion of *k-compatibility*. A pair of states is *k-compatible* if every input sequence *applicable* to the property specification, and of length k, produces the same output sequence when applied to the two machines

in those states. Two states are *compatible* if and only if they are k-compatible for all k. Therefore, verification starts out by finding all pairs of 1-compatible states and proceeding incrementally to find successive sets of k-compatible states (where $k = 2, 3, 4, ...$). When no further progress is possible, the algorithm would have computed all pairs of compatible states in the two machines. Based on this result, if the *starting state* of the property state machine is seen to have any compatible counterpart in the design state machine, then it is concluded that the design possesses the behavior expressed in the specification. If no such state is found, the design is declared devoid of the specified property.

Since property machines are allowed to have transitions with unspecified inputs, the requirement of compatibility is slightly modified by introducing the idea of *sequence sets*. Sequence sets are subsets obtained by dividing the set of all input sequences (applicable to the property machine in the state under consideration) such that the subsets are mutually disjoint. This may be done by enumerating all possible values of the inputs for transitions with unspecified inputs. For instance, considering the specification machine of Figure 4.4(b), for an input sequence of length 3, the sequence sets are: $\{1, 0, 0\}, \{1, 0, 1\}, \{1, 1, 0\}$, and $\{1, 1, 1\}$. With this extension, a pair of states in the design and specification machine are considered compatible if they produce identical output sequences for all input sequences in *at least one* sequence set.

The following section describes various steps involved in the verification procedure.

4.4.3. Verification Procedure

As discussed in Section 4.4.2, verification of properties involves finding compatible state pairs in the state machines corresponding to specification and design.

The first step is to find all 1-compatible state pairs, P_1, as the union of:

1. the set of state pairs having identical input-output behavior, where the *inputs are defined* for the specification
2. the set of state pairs producing identical outputs, where the *inputs are not defined* in the specification

The set of all 2-compatible pairs, P_2, is then found as the union of:

1. the set of 1-compatible pairs, *all* of whose successors are also 1-compatible, when the outgoing transition from the specification state is *fully* specified
2. the set of 1-compatible pairs, *some* of whose successors are also 1-compatible, when the outgoing transition from the specification state is *not fully* specified

This incremental procedure is repeated until the *fixed point* $P_{k+1} = P_k$ is reached. The set of *compatible states* is then given by $\mathcal{P}= P_k$.

To make the procedure efficient, states are handled in *sets* rather than individually. To do this, sets of states are represented by their *characteristic functions* (predicates on boolean state variables), and *next-state functions* are represented as boolean functions that map *sets of states* into *sets of states* .

For the following discussion, consider a Mealy type state machine, $M = (I, O, S, \delta, \lambda)$, where,

$$I = \{0,1\}^m \quad \text{is the input space,}$$

$$O = \{0,1\}^l \quad \text{is the output space,}$$

$$S = \{0,1\}^n \quad \text{is the state space,}$$

$$\delta : I \times S \rightarrow S \quad \text{is the next-state functional vector,}$$

$$\text{and} \quad \lambda : I \times S \rightarrow O \quad \text{is the output functional vector}$$

Also, let

$$\vec{q} = \text{the vector of present-state variables,}$$

$$\vec{x} = \text{the input vector,}$$

$$\vec{Q} = \text{the vector of next-state variables,}$$

$$\text{and,} \quad \vec{y} = \text{the output vector}$$

Thus, the next state vector Q_i and output vector y_j can be determined using the next state functional vector and output functional vector as:

$$Q_i = \delta_i(\vec{x}, \vec{q}), \quad i \in \{i...n\},$$

$$\text{and,} \quad y_j = \lambda_j(\vec{x}, \vec{q}), \quad j \in \{i...l\}$$

Let C_a be the partially specified specification machine and C_b, the corresponding design machine. U is the set of specification states having transitions with *unspecified* inputs, and $U(\vec{q_a})$ the *characteristic function* representing these states. Transitions

with *defined outputs* are represented by predicate η and those with *undefined outputs* by its negation, $\neg\eta$.

Verification (finding any design state compatible with start state of specification) is carried out in the following steps:

Step 1: Compute the set of transition pairs in the two machines having identical input-output behavior. This set,

- *excludes* transitions in the design with undefined outputs, given by $\neg\eta_{b_i}$ for the ith output
- *includes* transitions in the *specification* with undefined outputs (since they match all corresponding transitions in the design), given by $\neg\eta_{a_i}$ for the ith output

Considering all l outputs, these transition pairs are encoded by the predicate T:

$$T(\vec{x},\vec{q_a},\vec{q_b}) \;=\; \bigwedge_{i=1}^{l}\left[\Big(\big(\lambda_{a_i}(\vec{x},\vec{q_a})\odot\lambda_{b_i}(\vec{x},\vec{q_b})\big)\;\bigwedge\;\eta_{b_i}(\vec{x},\vec{q_b})\Big)\;\bigvee\;\neg\eta_{a_i}(\vec{x},\vec{q_a})\right]$$

Here, $\odot$ is the boolean *equivalence* (X-NOR) operator and the term $\lambda_{a_i}(\vec{x},\vec{q_a})\odot\lambda_{b_i}(\vec{x},\vec{q_b})$ represents matching outputs of the two machines.

Step 2: Using the predicate T found in Step 1, compute pairs of compatible states in the design and specification machines, starting with 1 compatible states. Noting that U is the set of specification states having transitions with unspecified inputs,

- the 1-compatible state pairs that include specification states in U satisfy T for *some* input values
- the 1-compatible state pairs that include specification states *not* in U satisfy T for *all* input values

The set of 1-compatible state pairs is then the union of these two disjoint sets and is given by the characteristic function:

$$P_1(\vec{q_a}, \vec{q_b}) \;=\; \Big(U(\vec{q_a}) \bigwedge \exists \vec{x} : T \Big) \;\bigvee\; \Big(\neg U(\vec{q_a}) \bigwedge \forall \vec{x} : T \Big)$$

Step 3: From the 1-compatible states found in Step 2, determine the 2 compatible, 3 compatible etc. states until no more reduction is possible. Given the set of k compatible states P_k, the set of $k+1$ compatible states P_{k+1} can be found by taking the *inverse image* of the set P_k, since, by definition, the *next states* of all states in P_{k+1} must be in P_k. (*inverse image* of a set of states is the set such that all the transitions from the latter reach the former). There are two inverse images to be considered:

- the inverse image of P_k such that *all* successor states are k compatible, given by, $\forall \vec{x} : P_k(\vec{Q_a}, \vec{Q_b})$; these correspond to states with all transitions having *specified* inputs
- the inverse image of P_k such that *some* successor states are k compatible, given by, $\exists \vec{x} : P_k(\vec{Q_a}, \vec{Q_b})$; these arise from states

having transitions with *unspecified* inputs. Also, in this case, such transitions must have identical outputs to satisfy the predicate T.

The union of these two sets (restricted to the set of k compatible pairs) gives the set of $k+1$ compatible state pairs:

$$P_{k+1}(\vec{q_a}, \vec{q_b}) \quad = \quad P_k(\vec{q_a}, \vec{q_b}) \bigwedge$$
$$\Big[\Big(U(\vec{q_a}) \quad \wedge \quad \exists \vec{x} : (T \wedge P_k(\vec{Q_a}, \vec{Q_b}))\Big) \bigvee \Big(\neg U(\vec{q_a}) \wedge \forall \vec{x} : P_k(\vec{Q_a}, \vec{Q_b})\Big)\Big]$$

Step 4: When the process in Step 3 reaches the *fixed point* (where no more reduction of set membership is possible), it gives the set of *compatible state pairs* $\mathcal{P}$.

ie., if $P_{k+1} = P_k$, then, $\mathcal{P}(\vec{q_a}, \vec{q_b}) = P_k(\vec{q_a}, \vec{q_b})$

Step 5: Once $\mathcal{P}$ is obtained, characteristic functions can be formed to give:

- the set of *design states* compatible with the *start state* of the specification, $\mathcal{P}(\vec{q_a}, \vec{q_b})_{\vec{q_a}=\vec{q_a}_{start}}$.
- the set of *design states* that are *not compatible* with *any* state of the specification, $\neg\exists \vec{q_a} : \mathcal{P}(\vec{q_a}, \vec{q_b})$.

Step 6: Depending on the outcome of Step 5 and the nature of properties to be verified (as expressed in the specification), it may now be determined whether the design possesses the specified behavior or not. This concludes the verification.

Based on the ideas presented in this and the previous chapters, the following chapter demonstrates how to model and verify authentication protocols, using as example the classic Needham-Schroeder protocol discussed in Section 2.3.2.3.

Chapter 5

MODELING AND VERIFICATION

State machines provide an excellent framework for modeling and analyzing authentication protocols. Though all the advantages of FSM based modeling to general communication protocols are also applicable in this case, there are certain issues that need particular attention. This chapter explores issues pertinent to modeling authentication protocols, and develops a verification methodology based on techniques discussed in Chapter 4. The classic Needham-Schroeder public key authentication protocol is used as an example throughout this chapter for discussions.

5.1. Verifying Needham-Schroeder Public Key Protocol

Needham-Schroeder Public Key protocol [39], described in Section 2.3.2.3, is an authentication protocol for large networks using public key cryptography. It aims to achieve mutual authentication of two parties with the help of a trusted third party (*certification authority*), and is one of the most widely studied authentication protocols.

There are two main reasons why this protocol is important. First, it is one of the earliest authentication protocols to use public key cryptography and is hence

considered a classic case study. Second, it is vulnerable to a subtle attack which went unnoticed in various studies and analyses for almost 17 years.

The attack on the protocol is described in Section 2.4.2.2. For ease of reference, a concise version of the protocol and the attack are reproduced in Figure 5.1 and Figure 5.2, respectively.

$$
\begin{array}{lcccl}
(m1) & A & \longrightarrow & B & : \ \{N_a, A\}_{K_b} \\
(m2) & B & \longrightarrow & A & : \ \{N_a, N_b\}_{K_a} \\
(m3) & A & \longrightarrow & B & : \ \{N_b\}_{K_b}
\end{array}
$$

Figure 5.1. Needham-Schroeder public key protocol

$$
\begin{array}{lcccl}
(m1) & A & \longrightarrow & Z & : \ \{N_a, A\}_{K_z} \\
(m1') & Z(A) & \longrightarrow & B & : \ \{N_a, A\}_{K_b} \\
(m2') & B & \longrightarrow & Z(A) & : \ \{N_a, N_b\}_{K_a} \\
(m2) & Z & \longrightarrow & A & : \ \{N_a, N_b\}_{K_a} \\
(m3) & A & \longrightarrow & Z & : \ \{N_b\}_{K_z} \\
(m3') & Z(A) & \longrightarrow & B & : \ \{N_b\}_{K_b}
\end{array}
$$

Figure 5.2. Attack on Needham-Schroeder public key protocol

In the protocol, A and B perform mutual authentication by demonstrating the knowledge of their *private keys*. Message exchange is secured by encryption with respective *public keys*. In the attack, a third entity Z (a malicious, but legitimate, principal in the system) cleverly manipulates messages to fool B into believing that Z is A.

Using this protocol as an example, this section develops a verification framework which models authentication protocols as interactive systems of *algorithmic state machines* representing communicating entities, and enables the verification of their authentication properties in an intuitive, yet rigorous and formal way.

5.1.1. Modeling the Authentication System

The protocol entities are modeled using state machines interacting with one another through protocol messages as outlined in Section 4.2. Therefore, the system can be viewed as a composite state machine resulting from these interacting component state machines.

Because of their flexibility and expressive power, algorithmic state machines (ASM) described in Section 4.3 are used to model the protocol entities. ASM models are very useful in authentication protocols because several data transformation operations (such as encryption and decryption) need to be represented concisely, a requirement not easily satisfied when using conventional FSM representations. The *control flow part* of these machines is then derived and used for formally verifying relevant properties. The *data processing part* may be separately analyzed to provide inputs to support control flow checking, and to validate vulnerabilities identified during verification.

The following types of entities can be identified as important in modeling the the original Needham-Schroeder public key protocol:

1. *Certification Authority (CA)*, the trusted third party from which entities in the system may obtain public keys of other entities.
2. *Initiator (I)*, which initiates an instance of the protocol by sending the first message.
3. *Responder (R)*, which is the target of the first message sent from an initiator.
4. *Intruder (Z)*, the malicious entity that tries to subvert the protocol. An intruder could be a legitimate entity in the system and may be able to participate in proper authentication procedures. In addition, it maybe able to observe all messages in the network, delete messages sent by others, alter messages in transit, or insert new messages. While participating in authentication sessions, it may also choose not to follow the rules stipulated by the protocol, and instead, resort to illegal manipulations or replays of messages. About the only limitation imposed is that, based on the *strong cryptography assumption*, an intruder will not be able to access the contents of encrypted messages for which it does not possess the decryption key.

5.1.2. Modeling Trusted Entities

Though in reality any user may assume the role of *initiator* or responder, it is convenient to assume (without loss of generality) that users are divided into disjoint sets of initiators and responders. Initiators and responders are assumed to adhere to protocol rules and message sequences, and are considered trustworthy.

Algorithmic state machines of an *initiator* and a *responder* in the Needham-Schroeder public key protocol (henceforth referred to as *NSPK protocol*) are shown in Figure 5.3 and Figure 5.4, respectively. In these figures,

$m1, m3, m3$	:	messages in the protocol
Ki, Kr	:	*public keys* of initiator, responder
Ki^{-1}, Kr^{-1}	:	*private keys* of initiator, responder
$E(m) : K$	:	encryption of message m with public key K
$D(m) : K^{-1}$	:	decryption of message m with private key K^{-1}
$nonce()$	:	generation of a fresh random number used as *nonce*
Ni, Nr	:	nonces generated by initiator, responder
$Tx(m)$	:	transmission of message m
$Rx(m)$	:	reception of message m
T, To	:	time-out mechanism to detect lost messages
end	:	end of an authenticated session
commit	:	final status of an authentication session

In the figures, $(I0, ..., I3)$ and $(R0, ..., R4)$ are the various states in which the initiator and responder can be. $I0, R0$ are the initial states and $I3, R4$ are the final commit states (states where an entity has accepted the authenticity of the other entity and has committed to a trusted session).

The basic state machines representing the control sequencing of initiator and responder can be derived in a straightforward manner from the algorithmic state machines, and are shown in Figure 5.5 and Figure 5.6, respectively.

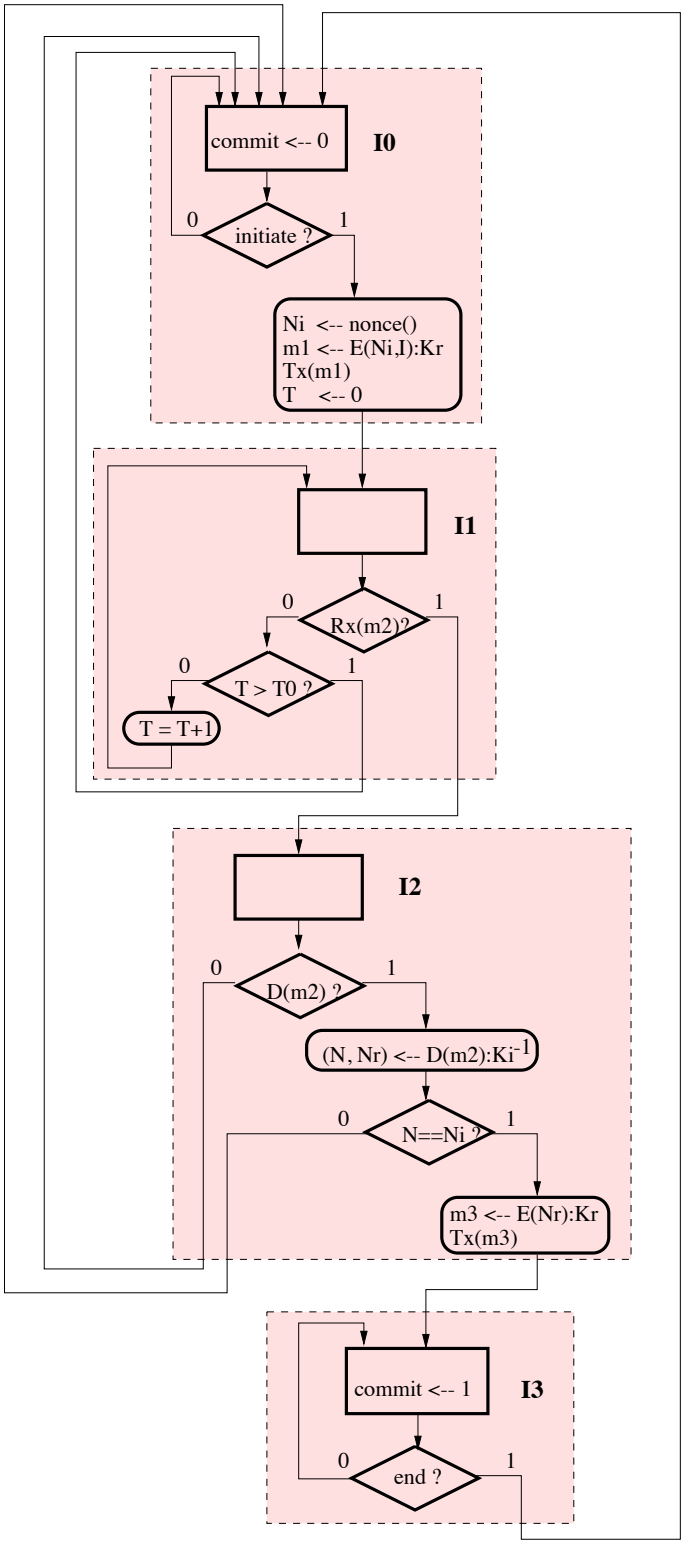


Figure 5.3. ASM of *initiator* in *NSPK protocol*

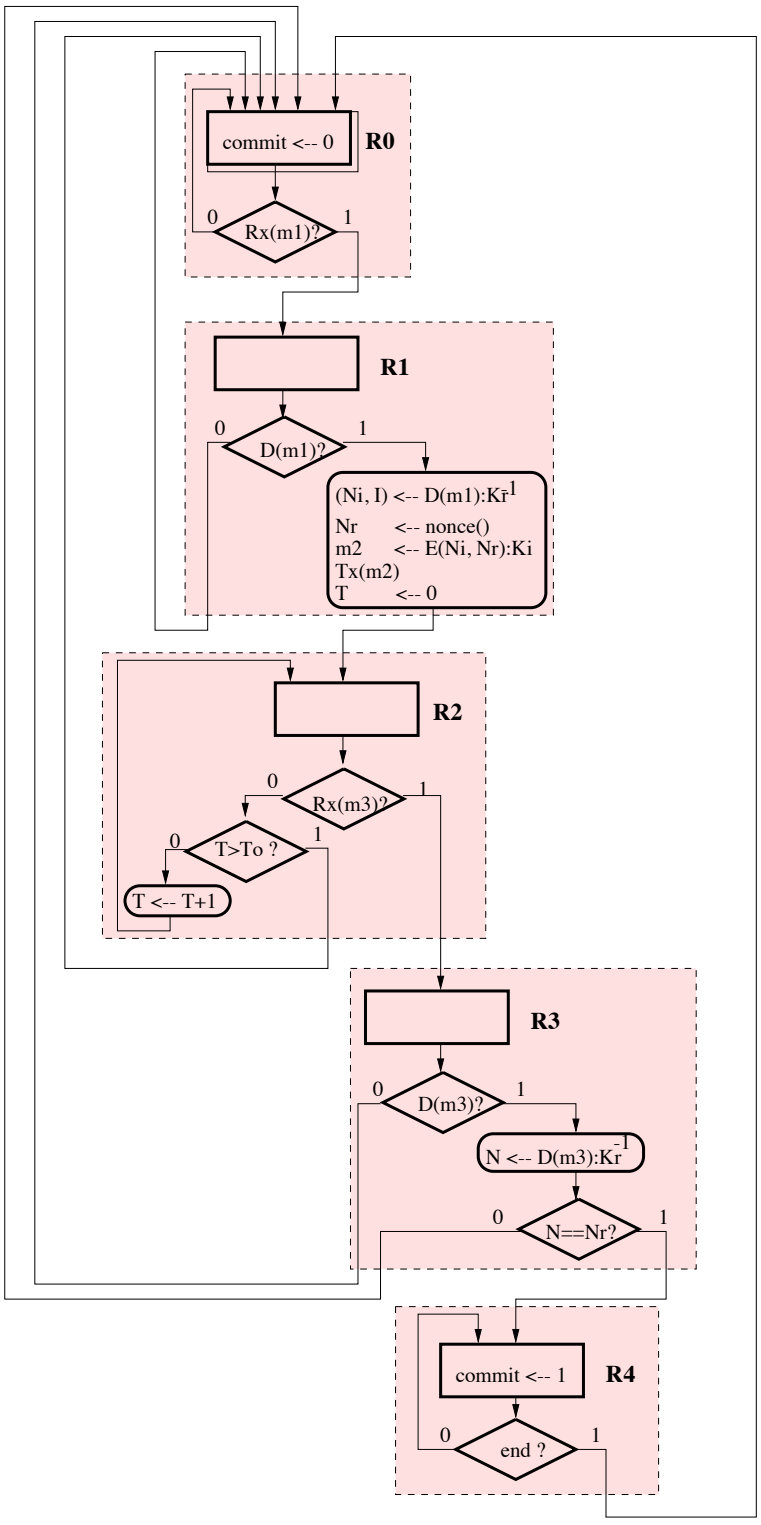


Figure 5.4. ASM of *responder* in *NSPK protocol*

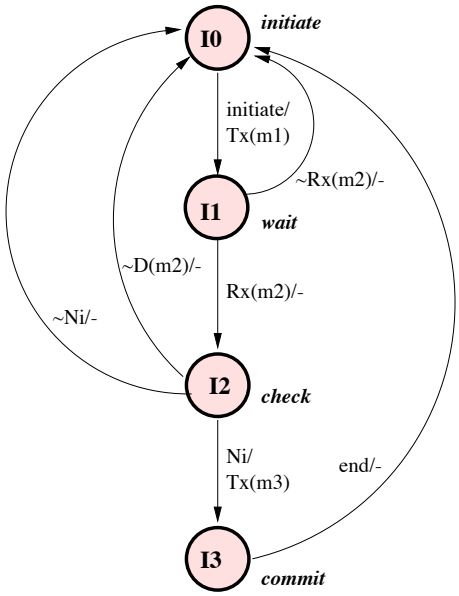


Figure 5.5. Control state machine of *initiator* in *NSPK protocol*

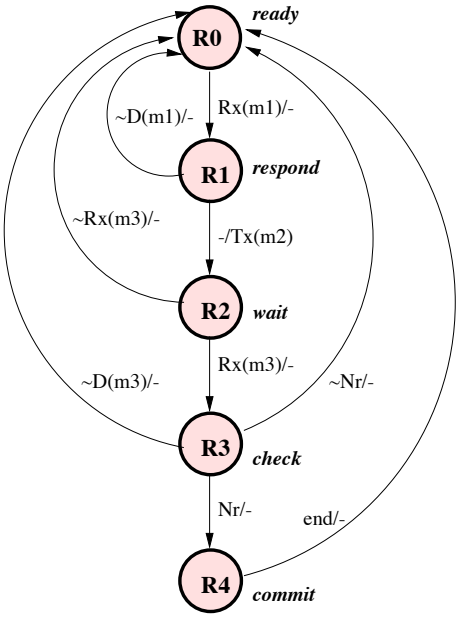


Figure 5.6. Control state machine of *responder* in *NSPK protocol*

5.1.3. Modeling an Intruder

Modeling an intruder is comparatively more complicated because of the variety of actions it may take at each step in the protocol, and the non-determinism in its choice. An intruder must be considered capable of performing one or more of the following operations:

- Initiate a session with any other entity and follow the protocol faithfully for proper authentication.
- Initiate a session with any other entity, but not follow the protocol properly and possibly do illegitimate operations such as manipulating messages, sending out-of-sequence messages, and replaying old messages, all with the intention of subverting the authentication mechanism.
- Observe any message on the network and possibly save it for later replay.
- Intercept any message sent in the network and:
  - Destroy the message.
  - Send the message unmodified to the original recipient.
  - Send an altered version of the message to the original recipient.
  - Replace it with an old valid message and send to the original recipient.
  - Send the message to someone other than the original recipient (possibly after alteration).
  - Use the message in another protocol run.

Further, due to the uncertainty in its motives and consequent actions, an intruder is assumed to be capable of acting in the role of either initiator or responder. As part of multiple simultaneous instances of the protocol with more than one entity,

it may potentially act in both roles. While doing so, the intruder may use messages from one instance of the protocol in the other.

These numerous possibilities make modeling of the intruder rather complicated. However, it may be done systematically (and to a degree of accuracy required for present purposes) by observing the following about the intruder's behavior:

- The only time the intruder acts *spontaneously* is when it initiates an authentication session with another user. This could involve a genuine message or a fake one, possibly a replay of an old valid message. The assumption is realistic because, except for $m1$, if any other message ($m2$ or $m3$) is randomly inserted, it will be ignored by other entities which follow protocol sequencing rules.
- At all other times, the intruder's actions are *reactive* , meaning, it becomes active on observing a message addressed to itself or to someone else. The resulting action could be one of the possibilities discussed earlier such as deleting the message, relaying the message modified or otherwise, replacing it with another, and redirecting it to someone other than the originally intended recipient.

The above observations lead to the algorithmic state machine shown in Figure 5.7, Figure 5.8, Figure 5.9, and Figure 5.10. The 'center of control' is in Figure 5.7, from where transfer of control occurs to other states in the remaining figures, and to where control eventually returns.

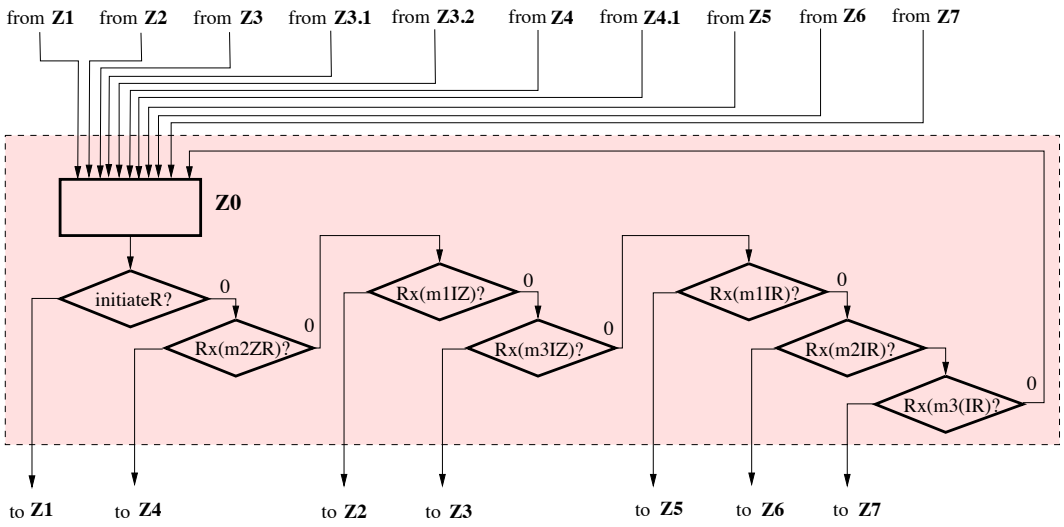


Figure 5.7. Part 1 of ASM of *intruder* in *NSPk protocol*

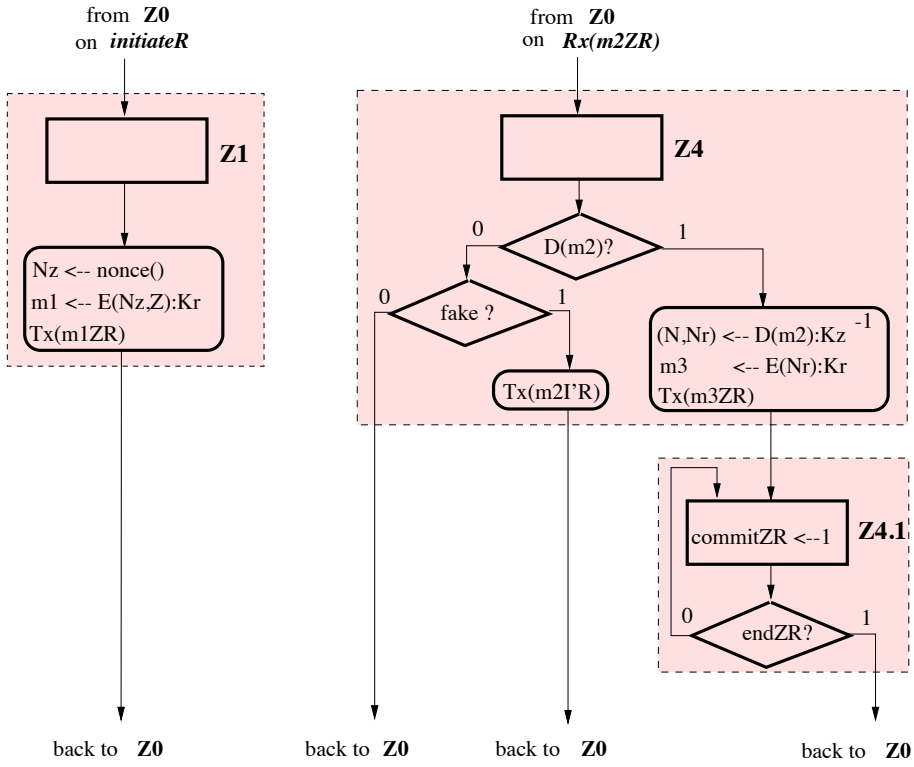


Figure 5.8. Part 2 of ASM of *intruder* in *NSPK protocol*

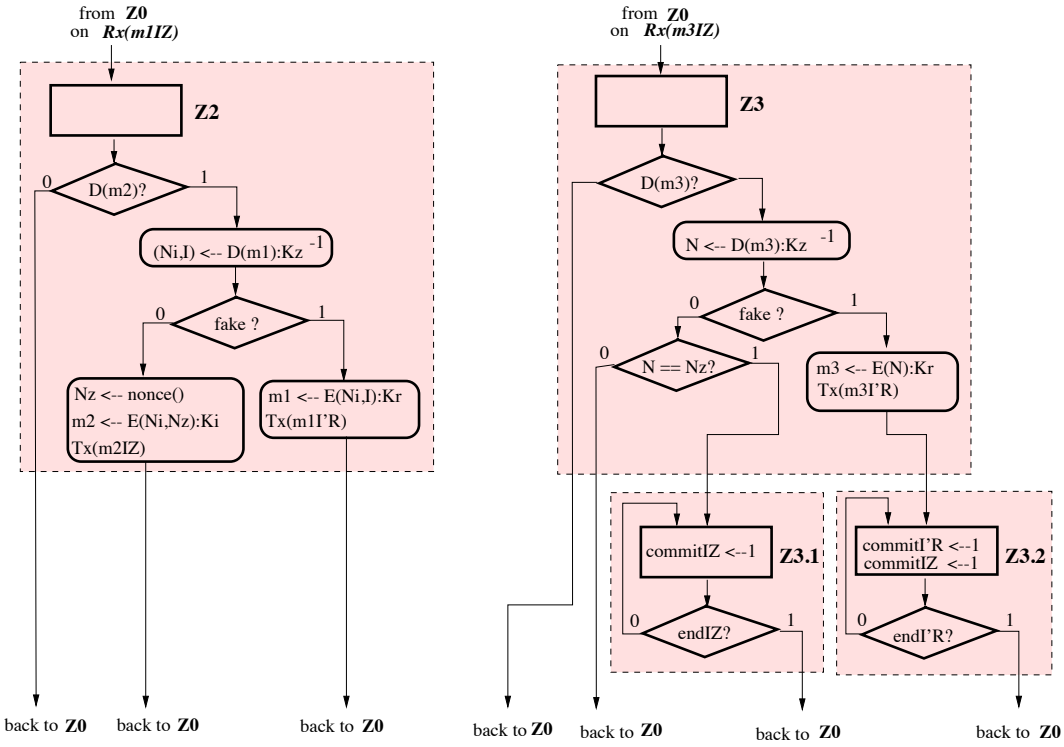


Figure 5.9. Part 3 of ASM of *intruder* in *NSPK protocol*

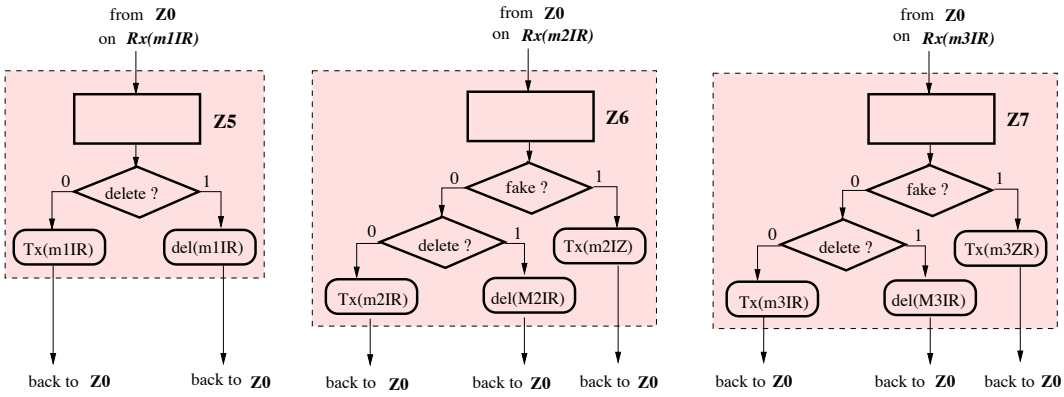


Figure 5.10. Part 4 of ASM of *intruder* in *NSPK protocol*

5.1.4. Modeling a Realistic Scenario

The simplest possible scenario for the Needham-Schroeder public key protocol involves an *initiator*, a *responder*, and an *intruder*. If this system is shown to be vulnerable to false authentication, then clearly any larger system with more entities will also have the same vulnerability. However, verifying that the simple system is secure does not imply that a larger system will remain secure under similar assumptions. To account for this, the system must be incrementally expanded to include more entities and the verification repeated until sufficient confidence is attained or a vulnerability is identified. In theory, even after analyzing a very large number of such cases, it may not be possible to conclude that a system is secure because of the new combinations introduced by incremental expansion. However, in practice, the analysis can often be safely limited to a convenient number of participants and protocol runs, as most authentication protocols generally have only a few number of steps, resulting in a limited number of possibilities. On the other hand, the case of a verification failure is strong, because any weakness found in a simple case will invariably appear in bigger systems anyway. In the case of Needham-Schroeder protocol considered here, a weakness is found in the simple scenario itself, allowing the verification procedure to conclude that the system is not secure, and making further incremental verification unnecessary.

The scenario modeled here comprises an *initiator* A, a *responder* B (both of which are trusted entities), and an *intruder* C (a legitimate entity that is malicious)

capable of participating in the protocol as initiator or responder. This model gives rise to the following authentication scenarios:

- A as initiator and B as responder
- A as initiator and C as responder
- C as initiator and B as responder

The initiator A is modeled with a state machine having states corresponding to its two possible authentication sessions (with B or C). This state machine, based on the basic initiator state machine (Figure 5.5), is shown in Figure 5.11.

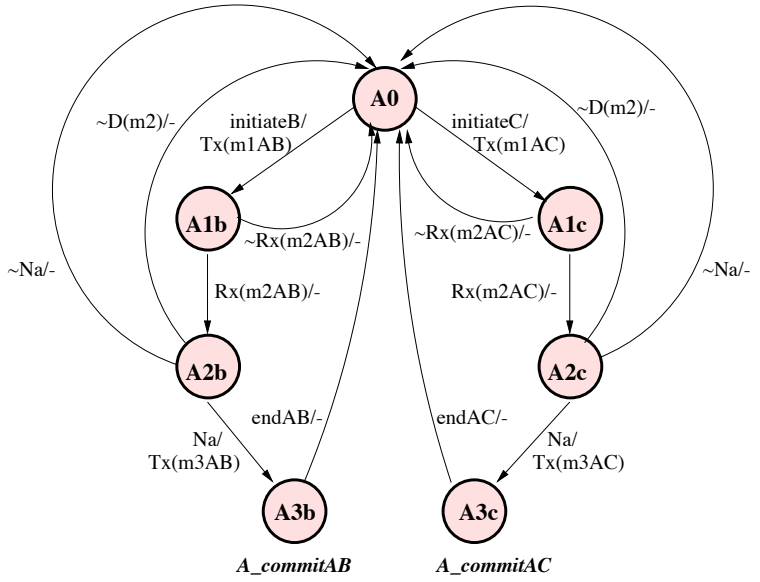


Figure 5.11. State machine for initiator A in NSPK protocol

Similarly, based on the responder state machine (Figure 5.6), B is modeled to include states corresponding to its possible sessions with A or C. Figure 5.12 shows this state machine.

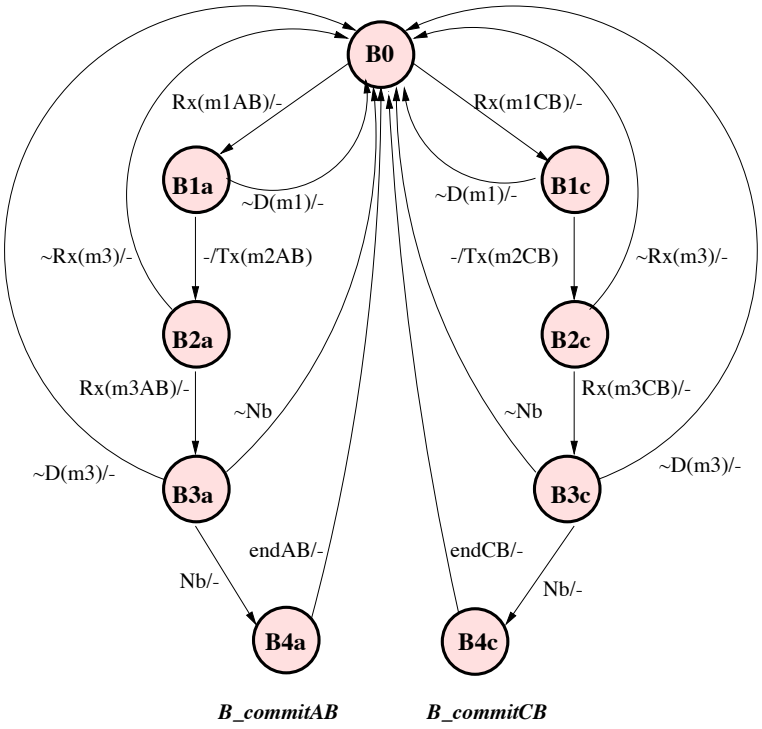


Figure 5.12. State machine for responder B in NSPK protocol

In these figures, a message of the form $mxPQ$ represents the xth message in the protocol run initiated from P to Q, a state label of the form PYq denotes the Yth state of P in its interaction with Q, and $P\_commitQP$ denotes establishment of P's belief in the authenticity of the session initiated by Q (meaning, P trusts that it is indeed Q at the other end). Termination of an authenticated session between P and Q is indicated by $endPQ$. Further communication must involve a new run of the protocol. A transition labeled $\sim Rx(m)$ (non-reception of a message) is caused by the loss of message m. Since the primary concern is with disruptions caused by the intruder, it is implied that a transition of this type results when the intruder destroys message m in the network. Similarly, $\sim D(m)$ denotes a message that

cannot be decoded by the recipient, possibly implying an alteration by the intruder. An example would be an encrypted message originally sent to someone other than the recipient, or a valid message that has been tampered with during transit. Transitions associated with $\sim N$ (non-matching nonce) imply authentication failure, possibly as a result of receiving an old valid message replayed by the intruder.

The entity C may participate in legitimate protocol runs as an initiator (while communicating with B) or as a responder (while communicating with A). In addition, it may also act maliciously in protocol runs between A and B as a passive observer or as an active intruder, performing various actions identified earlier.

A basic state machine describing intruder C's behavior is shown in Figure 5.13. Please note that, some actions that would obviously be detected and rejected by other principals have been left out to simplify the state machine. An example would be that of sending a message corresponding to the second step in the protocol to a principal who is in its initial state and does not have any ongoing sessions.

As seen in the diagram, most transitions in the intruder state machine are controlled by inputs n, d, or f. These inputs denote, respectively, that the intruder has decided to send a *normal* response as per the protocol, *delete* a message without the intended recipient seeing it, or send a *fake* message with the possible intention of masquerading as another entity. The states annotated with $C\_commitAC$ and $C\_commitCB$ are states where authentication involving C, acting in its real identity, has succeeded. C may also try to pass off as A and send fake messages to B as if coming from A, as indicated by $m1A'B$ etc. In such a scenario, state $C\_commitA'B$

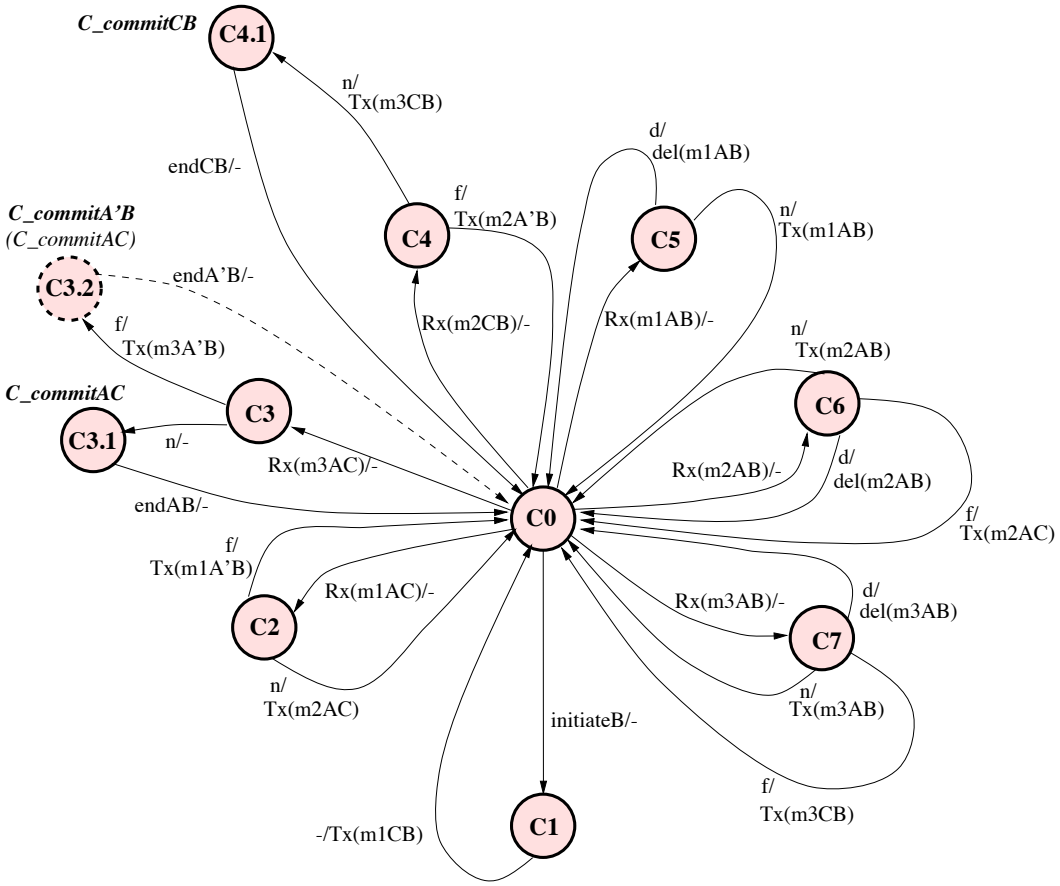


Figure 5.13. State machine for intruder C in NSPK protocol

may potentially be reached, meaning, C has falsely convinced B that it is actually A. If this state can indeed be reached, then C has successfully broken the protocol.

5.1.5. Verifying Authentication

A verification framework with associated tools for design verification based on the technique of using *property state machines* as outlined in Section 4.4 has been developed at the University of Texas at Austin [24]. This framework can be used for the verification of authentication protocols as well, because most communication

protocols can be approximated as synchronous state machines. In the case of authentication protocols, the *design state machine* is the composite state machine obtained from the interaction of the state machines corresponding to *initiators*, *responders*, and *intruders*.

Continuing with our example, the Needham-Schroeder authentication protocol is verified using the *design state machine* developed in the previous section and the verification methodology based on *property machines*. This is done by specifying the criterion for the success or failure of authentication as a *property state machine (specification machine)* and verifying it against the *design state machine*. Essentially, the verifier checks whether there is *any state in the design state machine* that is *compatible* with the *start state of the specification state machine* . The presence of any compatible state implies that the design possesses the behavior described by the specification state machine.

As explained in Section 4.4.1, the type of the property chosen for verification has important implications. Here 'type' refers to the classification of the property as a *good property* or a *bad property*. For an authentication protocol, a *good property* would be that proper authentication is achieved between two trusted principals, whereas a *bad property* would be that an intruder achieves false authentication by cheating another entity. Since the protocol is modeled in a minimal scenario comprising only one of each type of entity (initiator, responder, and intruder), the only verification that can be considered conclusive is the case where the *bad property* is shown to hold true. This is because, if the protocol fails in the minimal scenario, then it is certain

to fail in larger systems with more entities. On the other hand, verifying that the bad behavior is absent in the minimal system (meaning, there is no false authentication) cannot be considered conclusive because the bad behavior may manifest in a bigger system with more entities and consequently a larger number of potential interactions. By the same argument, verifying that the good property holds is also not conclusive, because the property may not hold good in a larger system. Therefore, when studying authentication properties (especially if the system is modeled in a minimal configuration), it is the presence of *bad behavior* that should be verified.

The scenario described in Section 5.1.4 can be checked for a *bad property*, meaning, whether the intruder can achieve false authentication. This is done by specifying a corresponding property state machine having a start state where both the initiator A and responder B are in initial states $A0$ and $B0$, and an end state where they have conflicting *commit status*. For instance, a false authentication could lead B to believe that it is communicating with A, when there may not even be a session between A and B. In such a case, the end state of the property state machine will have B in '$B\_commitAB$' whereas A will not be in '$A\_commitAB$'. The intermediate states and transitions are left undefined because the verification procedure will search for any path between the given start and end states and satisfying the property. If the composite design state machine is found to have a state which is compatible with the start state of the property state machine, the presence of a flaw is evident. Figure 5.14 shows one way of specifying the property state machine.

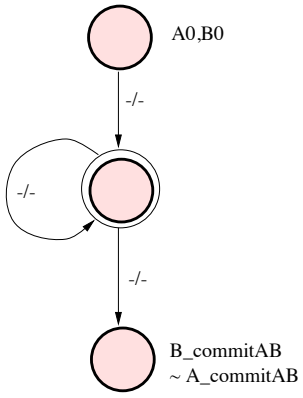


Figure 5.14. A property state machine for verifying the NSPK protocol

The state machine specifies the property that, starting from initial state, the system ends up in a state where the commit status of A and B are contradictory - B is in '$B\_commitAB$' whereas A is *not* in '$A\_commitAB$'. Intermediate states are left unspecified so as to identity *any* matching state. A match for the property, if found in the design, indicates the presence of a flaw in the authentication process.

The property is checked against the design state machine of the system, which has the product state space of the state machines of initiator (A), responder (B), and intruder (C). In the modeled scenario, verification reveals a flawed authentication path as shown in Figure 5.15. The figure indicates the states through which the system passes as a result of the message sequence manipulated by the intruder.

Since the intermediate transitions and outputs in the property state machine are left unspecified, an exhaustive search for all matching paths is performed. By specifying any known input or output values, the search space can be cut down, making the search more efficient. One such modification is to specify the input f as *true* (indicating that C may send fake messages). This is reasonable because only the

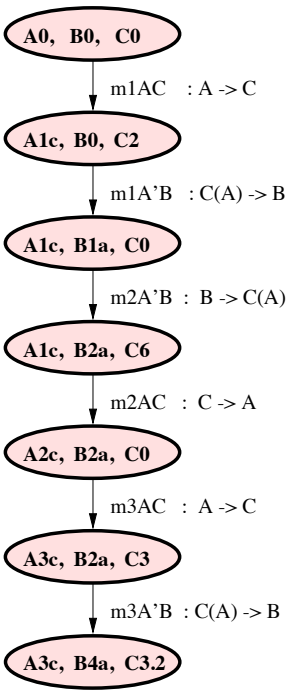


Figure 5.15. Flaw in NSPK protocol, uncovered in verification

malicious activities of the intruder are of interest. This added constraint limits the verification procedure to only those transitions corresponding to the *true* value of the f input and saves considerable computation by avoiding many matching steps which would otherwise be carried out. However, it is important to note that in both cases the flaw will be found eventually, but with possibly many more steps when inputs are unspecified.

Had this verification on the minimal system failed to find a weakness, it would be necessary to repeat the verification procedure by incrementally introducing more entities until a sufficient level of confidence is achieved. The cut off point for such a repetitious process is very much dependent on the type of protocol being verified and the various possibilities of entity interaction that may arise.

This demonstrates the application of the verification technique, based on ASM modeling and property verification principles, to authentication protocols. The procedure is generic enough to be applicable to a wide variety of protocols. Also, being capable of uncovering the subtle flaw in the Needham-Schroeder protocol, it is not difficult to use the framework to detect other types of protocol weaknesses (many of which, incidentally, are much less complicated). Thus, it is hoped that the framework would be a potentially valuable tool in modeling and analyzing authentication protocols.

Chapter 6

SIMULATION BASED VALIDATION

Chapter 5 discussed a technique for formally verifying authentication protocols. Though powerful, application of verification techniques is complex and often demands considerable expertise. Also, the complexity of modeling and the demands on computational resources increase rapidly as the size of the protocol being verified increases. Thus, it is generally difficult to apply formal verification procedures to complex protocols. Therefore, an alternative approach based on *simulation* is investigated in this chapter. Though lacking the rigor of formal verification, simulation can generally give good results to help validate authentication protocols.

6.1. Protocol Validation Using Simulation

Discrete event simulation is used extensively in the study of communication networks and protocols [33] [45] [46]. By injecting randomly generated events into a model of the system under study, simulation helps to observe the system's response under various conditions. Such a study may begin by simulating a rudimentary model of the system, and then incrementally refine the model to obtain increasingly accurate and realistic results. Validity of the results depends largely on the accuracy of the

model, the coverage of various events and event sequences during the simulation, and the duration of simulation. Simulation lacks the rigor of formal verification, but is capable of producing close enough results if designed and carried out properly.

This chapter investigates the application of simulation techniques in validating authentication protocols. Because of the randomness involved in simulation, caution should be exercised in deriving conclusions based on the results of a simulation study. In particular, if simulation does not reveal any weakness in an authentication protocol, it can only be inferred that the protocol is very *likely* to not have any flaw, but it cannot be concluded that it is *certainly* free of flaws. On the other hand, if simulation does reveal a weakness, it can be safely concluded that the protocol is flawed.

The remainder of the chapter details the application of simulation to authentication protocols, using the example of *Needham-Schroeder public key protocol*, the classic bench mark protocol that was described in Section 2.3.2.3.

The simulation study was performed using *Opnet*, a sophisticated simulation tool widely used in the study of communication protocols. The following section provides a brief overview of the tool and its capabilities.

6.2. Opnet Simulation Tool

Opnet (*Op*timized *N*etwork *E*ngineering *T*ools) [28] [34] provides a powerful discrete event simulation environment for modeling and simulation of a wide variety of communication protocols and systems. The system under study is modeled as

a hierarchy of abstract *levels (layers)*. The uppermost layer, *network model level*, captures the topological representation of the system as a network of communicating *nodes* and associated interconnecting *links*. The structure of each node in the network layer is defined in greater detail by the next lower layer in the hierarchy, the *node model level*. At this level of abstraction, each node is modeled with basic communication components such as *processors*, *queues*, *transmitters*, and *receivers*. The lowest layer, the *process model level*, defines the process behavior of modeled entities (nodes) through *state transition models* embedded within their processors and queues. Simulation triggers in the form of messages are injected by *packet generators* that can be configured to generate messages in flexible formats and following a variety of statistical distributions. Simulation results are gathered using configurable *probes* that monitor various elements in the model. These abstractions, together with user definable message formats and powerful communication primitives, offer a very flexible and powerful simulation environment.

Using Needham-Schroeder public key protocol as example, the following sections develop a framework to perform simulation studies on authentication protocols.

6.3. Simulation Model for Needham-Schroeder Public Key Protocol

A network of communicating entities that use the Needham-Schroeder public key authentication protocol (henceforth referred to as NSPK protocol) can be modeled in Opnet using the primitives mentioned in Section 6.2. In the modeled configuration,

there is one each of trusted *initiator* and trusted *responder*, and one *intruder* (a malicious entity). Being itself a possibly legitimate entity in the system, the intruder also possesses the functionality of both initiator and responder.

The ability of the intruder to observe and potentially manipulate any message in the network is modeled by forcing all messages to go through the intruder node as if it were acting as a transparent *relay* between communicating entities. This is a fairly valid assumption in many modern communication networks (for example, a *message router* could fall into this category). Besides merely passing the messages exchanged between other entities through, the malicious node may try to delete, modify, or manipulate them.

In the model, *initiator* nodes invoke authentication sessions with randomly chosen *responder* nodes at random intervals. It is to be noted that an initiator node may engage in sessions with the *intruder* node also because the latter is capable of functioning as a normal responder node (since it may be a legitimate principal in the system). Similarly, in its capacity as an initiator, the intruder node may engage in sessions with other normal responder nodes.

Figure 6.1 shows the scenario implemented as an Opnet *network level* model. In the figure, *i_101* and *i_102* are initiators, *r_201* and *r_202* are responders, and *z_301* is the intruder (the numerical parts in these names have no significance other than being convenient). A minimal configuration of the modeled system can be studied by activating only *i_101, r_201, and z_301* (ie., one each of initiator, responder, and intruder).

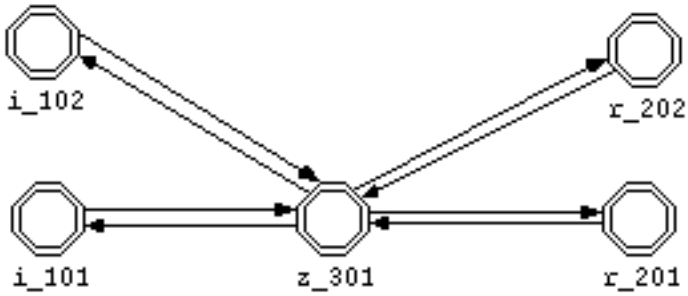


Figure 6.1. *Opnet network level model* of NSPK protocol authentication scenario

The internal structure of an initiator node is defined in the lower *node level* model shown in Figure 6.2. It essentially consists of a *processor* embedding the initiator's protocol behavior, a *transmitter*, a *receiver*, and communication *streams* connecting them. Transmitters and receivers attach to communication links between nodes. Similarly, Figure 6.3, and Figure 6.4 show the structures of a responder node and the intruder node, respectively. Note that the intruder node has multiple receivers and transmitter since it has links to both initiators and responders.

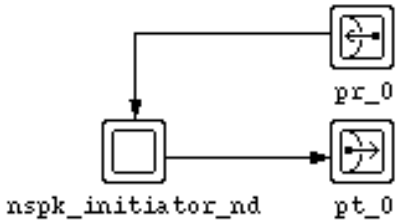


Figure 6.2. *Opnet node level model* of *initiator* in NSPK protocol

The protocol behavior of nodes is defined by *processes* running within their processors, which are implemented at the *process level* of Opnet as state transition diagrams and associated embedded code. Opnet allows executable code to be as-

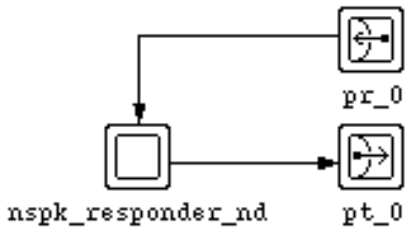


Figure 6.3. *Opnet node level model* of *responder* in NSPK protocol

sociated with states themselves in the form of *entry/exit executives*, or with state transitions as *transition executives*, or combinations of both. State transition models embedded within nodes corresponding to initiator, responder, and intruder are shown in Figure 6.5, Figure 6.6, and Figure 6.6, respectively. The process models are based on the FSM representations of these entities developed in Section 5.1.1. The state machines of initiators and responders are straight translations of the protocol rules and message sequences. However, the state machine of the intruder is considerably more complex as it needs to emulate the behaviors of both initiator and responder (since the intruder may act in either role), and various malicious possibilities including deletion, modification, and manipulation of messages.

6.4. Simulation Setup

In the modeled configuration, there are three possibilities for authentication sessions. This is because the intruder *z_301* is also considered to be a legitimate (but malicious) entity capable of acting either as an initiator or as a responder.

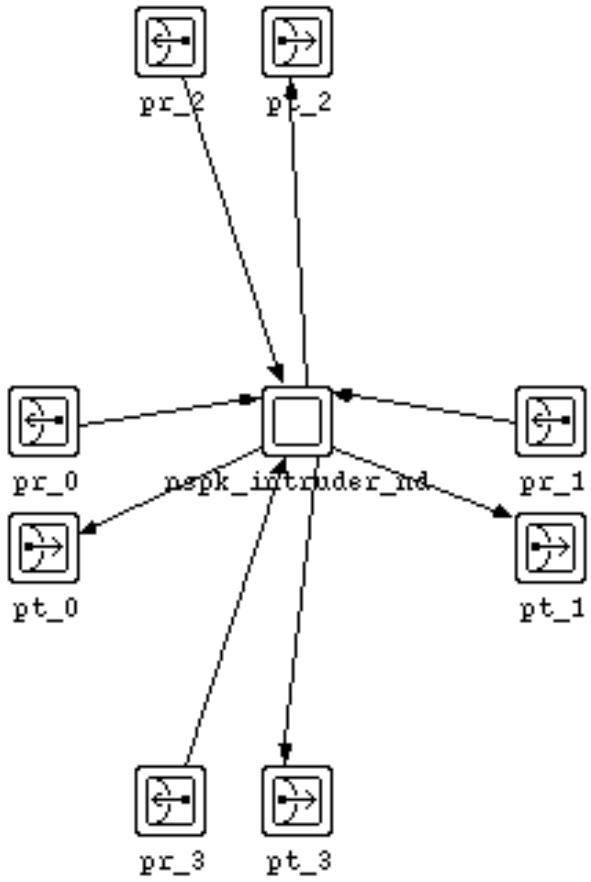


Figure 6.4. *Opnet node level model* of *intruder* in NSPK protocol

Sessions could exist between *i_101* and *r_201* (a trusted initiator and a trusted responder), between *i_101* and *z_301* (a trusted initiator and an intruder in the role of responder), or between *z_301* and *r_201* (an intruder in the role of initiator and a trusted responder).

In the simulation experiments, initiator *i_101* and intruder *z_301* attempt to initiate authentication sessions at randomly chosen instants, with a mean interval of about 500 simulation seconds. A successfully authenticated session is assumed to be active for 20 simulation seconds. Note that the time durations of 500 and 20 are

chosen to simplify simulation design (such as, to help avoid race conditions) and do not have significance otherwise; they do not result in any loss of generality. Messages sent between initiator *i_101* and responder *r_201* pass through intruder *z_301* as stated before. The intruder may relay, delete or manipulate the messages that are passing through, the exact nature of the action at any instant being chosen randomly. If all messages of a session are handled faithfully (ie., relayed transparently), the session succeeds, provided ofcourse that the authentication requirements are met. If any message is deleted or otherwise modified by the intruder, the authentication protocol is expected to enable the involved entities to detect such attempts and safely abort the session. However, there could be cases where some messages are manipulated by the intruder, and yet satisfy all the requirement of the protocol. Since the affected entities will not be able to detect such messages as malicious, these actions by the intruder may go undetected and eventually lead to false authentication.

In the configuration studied here, a false authentication would occur if the intruder *z_301* is able to pass off as the initiator *i_101* in a session with the responder *r_201*, or as the responder *r_201* in a session with the initiator *i_101*.

6.5. Simulation Results

The setup as described in the previous section was run for typical durations of 100,000 simulation seconds (averaging about 22 seconds real-time in a multi-user workstation running Digital Unix on a 300 MHz Alpha platform). At random intervals

(with a mean of approximately 500 simulation seconds), authentication sessions were attempted by initiator *i_101* (to responder *r_201 or to intruder z_301* acting in the role of a normal responder) and intruder *z_301* (to responder *r_201*). An average of about 290 authentication attempts were made in each run. The intruder randomly chose to relay faithfully, delete, or maliciously manipulate messages passing through it. In approximately 30 authentication sessions of each run, the intruder attempted faking, meaning, trying to pass of as initiator *i_101* in a session with responder *r_201*. The results showed that, in each run of simulation, an average of 3 such attempts succeeded. In those sessions with the responder *r_201*, the intruder *z_301* succeeded in falsely taking on the identity of the trusted initiator *i_101*. Message traces produced during the simulation indicate that the fake sessions succeeded precisely because of the flaw in the Needham-Schroeder public key protocol that was described in Section 2.4.2.2.

In this case, simulation was able to reveal the flaw by modeling a minimal scenario. Had the minimal configuration failed to expose any flaw, it could point to two possibilities. First, the simulation might not have run long enough to encounter the event sequences leading to the flaw. Second, the minimal configuration might indeed be free of flaws. The first possibility can be investigated further by running the simulation for incrementally longer durations until a weakness is revealed or a sufficient level of confidence has been achieved. However, it is not easy to define the threshold for this cut-off point. Even when sufficient confidence is gained in the minimal configuration, it cannot be taken as an assurance that a weakness may

not manifest in a larger configuration where more entities are involved. Because of this, the modeled system may have to be expanded incrementally by introducing more entities until a sufficient level of confidence is attained. Again, it is hard to clearly fix the cut-off threshold for such repetitions. However, considering that most authentication protocols generally involve only a few entities in each run, and have only a limited number of steps (messages), it may be possible to safely restrict the scope of incremental simulation to a practical level. In the present experiment, since the flaw is exposed in the minimal configuration itself, it can be concluded that the protocol is flawed, there is no need for further incremental simulation.

From the foregoing discussions and simulation results, simulation appears to be a potentially valuable validation technique for authentication protocols.

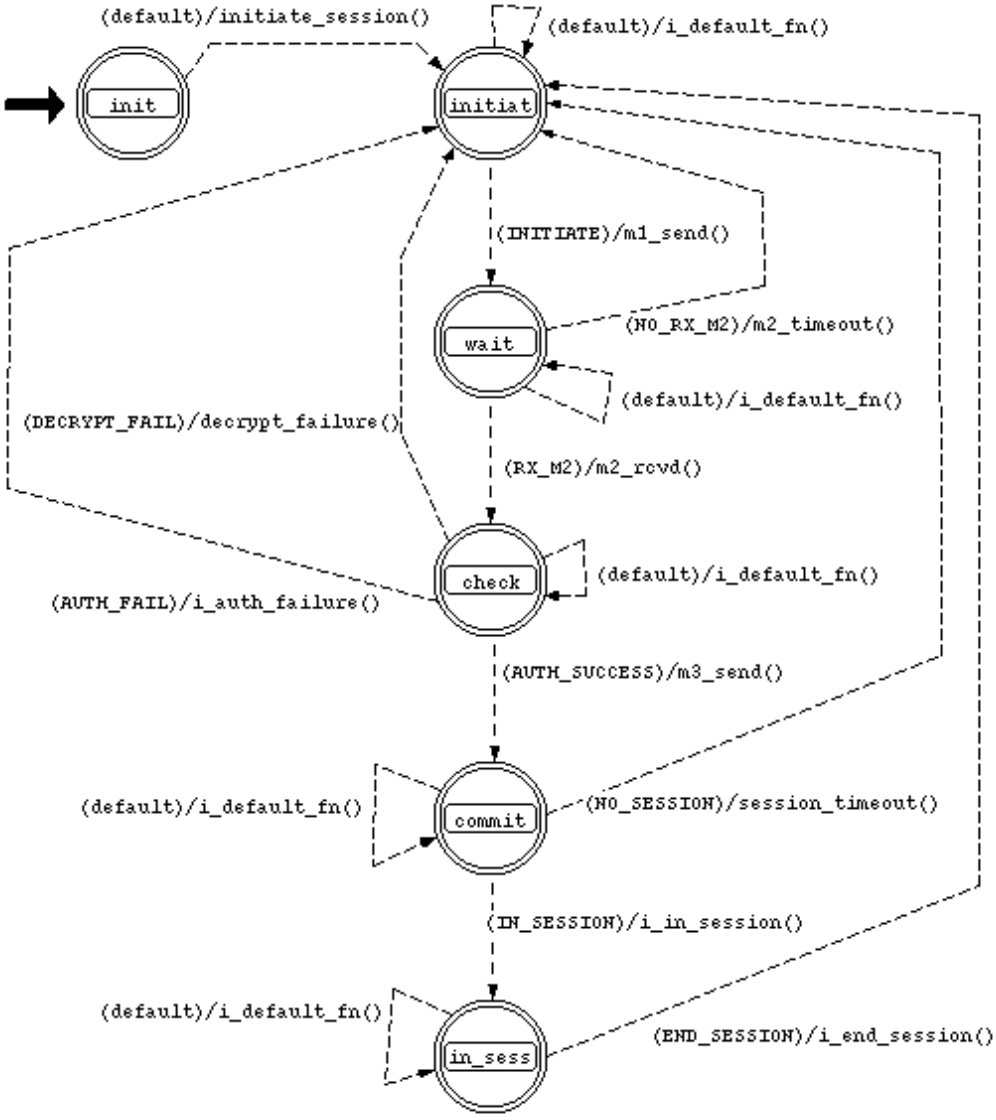


Figure 6.5. *Opnet process level model* of *initiator* in NSPK protocol

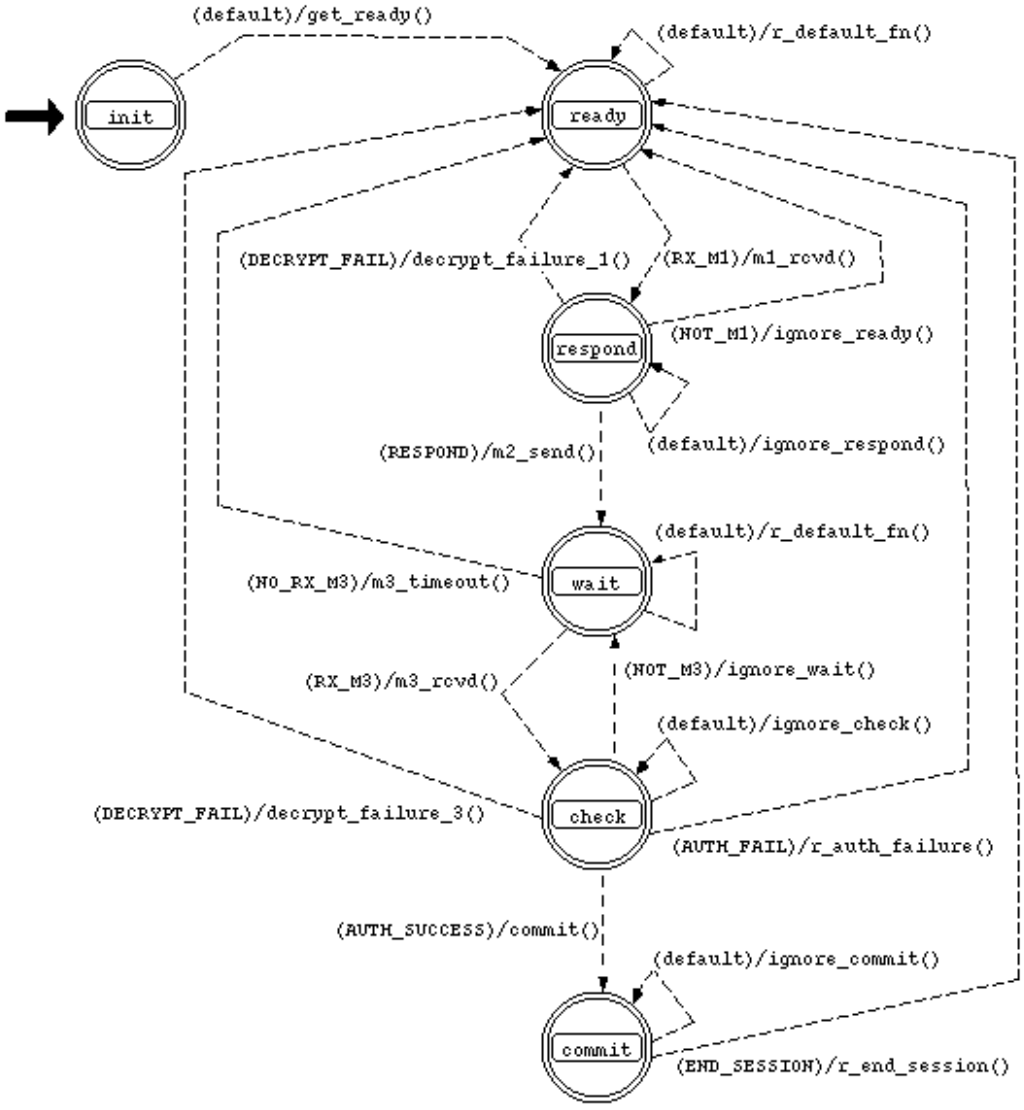


Figure 6.6. *Opnet process level model* of *responder* in NSPK protocol

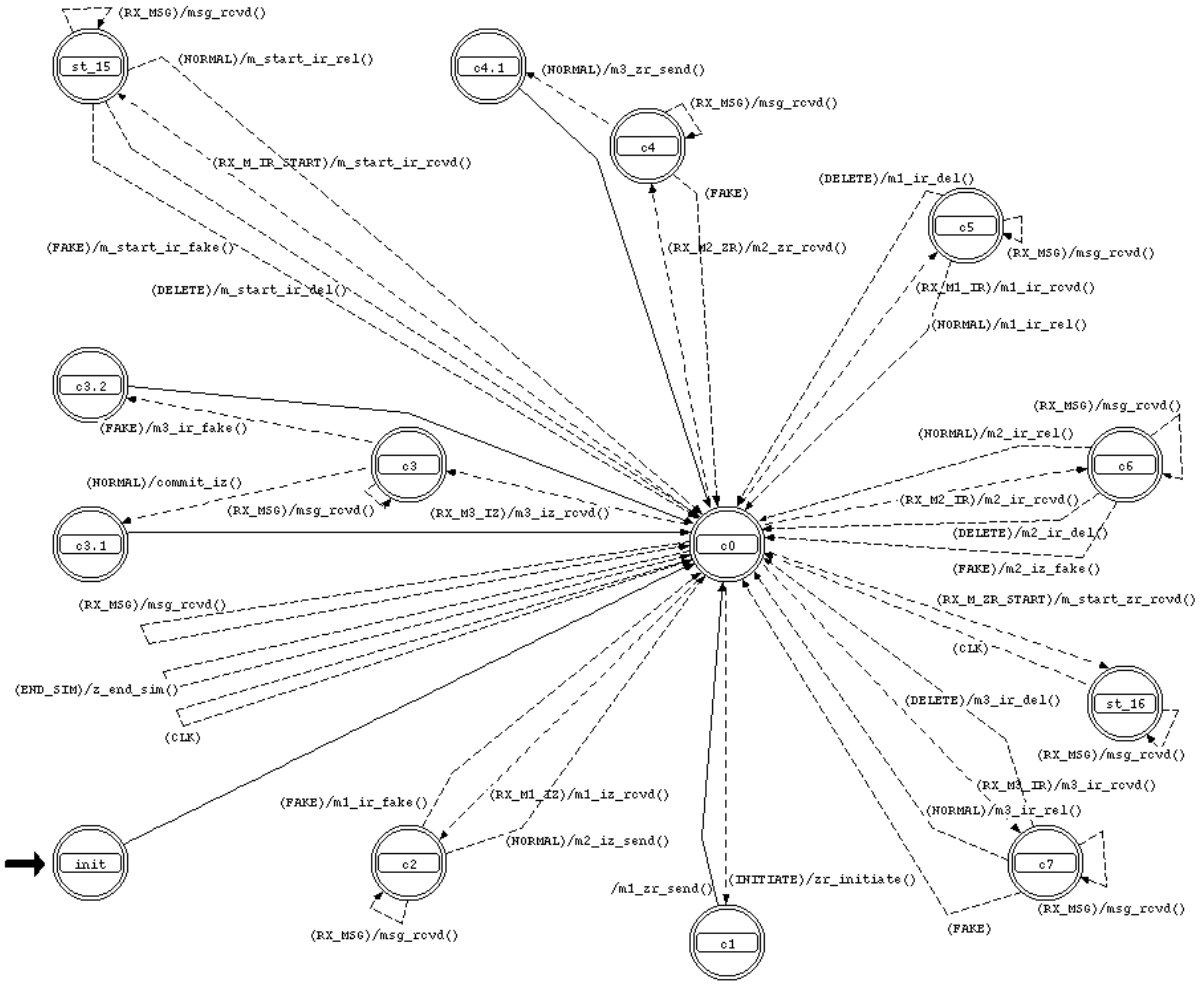


Figure 6.7. *Opnet process level model* of *intruder* in NSPK protocol

Chapter 7

META AUTHENTICATION

The goals of authentication protocols, their vital role in establishing secure communication channels, and some of their vulnerabilities were discussed in Chapter 2. Despite the simplicity of their structure and intuitiveness of operational logic, many authentication protocols have been found to be flawed in subtle ways. A malicious entity with sufficient resources and intelligence may potentially be able to identify and exploit these weaknesses, thereby defeating the authentication mechanism to masquerade as another valid entity. Though considerable research has been done in the past on various techniques to formally verify authentication protocols (Chapter 3), it is generally agreed that proving an authentication protocol secure is not easy or straightforward. As the role of networks in day to day activities continues to expand rapidly, particularly in electronic commerce transactions, this area of research still remains important and active. However, as protocols get increasingly complex, it is becoming considerably more difficult and cumbersome to subject them to verification procedures.

As Abadi and Needham rightly point out in [1], there is no substitute for good engineering practices in designing cryptographic protocols. Approaching the problem

from this direction, this chapter develops an architecture designed to prevent generally known attacks on authentication protocols. Attention has been paid to keep the architecture simple enough so that it will not face the same problems that the protocols it tries to protect face, yet robust enough to resist most forms of attack.

7.1. Meta Authentication

The term *meta authentication* denotes an 'encapsulating authentication' mechanism for general authentication protocols. This is achieved through a high level mechanism for validating the execution of underlying authentication protocols.

Meta authentication operates in the context of a *meta authentication framework* comprising an architecture and a high level validation protocol that together provide a distributed environment for monitoring and validating the execution of authentication protocols. By utilizing the services of this layer, entities involved in an authentication session can ensure that the execution of the authentication protocol itself has been proper and devoid of any malicious tampering or manipulation. The framework does not make any assumptions on the underlying authentication protocol and is generic enough to support any protocol chosen by communicating entities. It is important to note that, it is not a goal of meta authentication to offer any assurance as to the correctness of beliefs established by the 'encapsulated' protocol; this is still the responsibility of the encapsulated protocol. However, it does provide assurance that the encapsulated protocol itself runs correctly, protected from extraneous interference.

The remainder of the chapter is devoted to developing the meta authentication framework and to demonstrating how it helps to prevent various attacks.

7.1.1. Trust Model in Meta Authentication

Meta authentication is based on the concept of *trusted third parties*. All communicating entities in a *domain of trust* utilize the services of the trusted entity. The use of trusted entities is not new in authentication, several authentication protocols that depend on trusted third parties have been proposed and used in the past (a familiar example is the *Kerberos* authentication system [41]). However, in almost all such protocols, the trusted third party is an integral part of the authentication process. Any weakness of the trusted party can seriously damage the security of numerous entities depending on its service. Moreover, in large systems, the trusted entity can quickly become a performance bottleneck, as it needs to be directly involved in all sessions initiated among the entities in its authentication domain. Inter-domain authentication (between two entities belonging to two different domains) is also a major issue in such protocols, because of the potentially limited trust that entities in one domain may be willing to have on the trusted entity belonging to another domain.

In the scheme developed here, the role of trusted third parties is limited to only helping to monitor the execution of authentication sessions. They are neither directly involved in the execution of the protocol, nor do they play any role in establishing beliefs between communicating parties. Their service is needed only if the

communicating entities wish to protect their authentication process through meta authentication. Even without meta authentication, they will still be able to operate any authentication protocol as normally done. However, doing so could expose the authentication session to a variety of attacks as discussed in Chapter 2. The optional use of meta authentication is designed to protect against such attacks, while incurring minimal overhead.

In meta authentication, the entire user space is divided into *trust domains*. Each domain includes any number of ordinary communicating entities and a trusted *meta authentication server* (henceforth called the *meta server*). A meta server establishes trust with every member in its domain, so as to function as an intermediary in intra-domain authentication. An entity need not trust any member even in its own domain, other than its meta server. Moreover, entities in one domain need not trust the meta server in another domain. However, meta servers in different domains may establish and maintain trust on one another, so that their services can be extended to inter-domain authentication as well. As there is only one meta server in each domain, this trust is far easier to establish and manage as compared to maintaining trust among all entities in all domains, or between entities in one domain and meta servers in other domains, or even between all entities in the same domain. Essentially, meta authentication follows a hierarchical and transitive trust model. This means that, if there is trust between entity A and its meta server S_A, between entity B (possibly in another domain) and its meta server S_B, and between meta servers S_A and S_B, then eventually trust may be established between A and B.

Based on this concept, the following section describes an architecture capable of protecting both *intra-domain* and *inter-domain* authentication sessions.

7.2. An Architecture for Meta Authentication

The meta authentication scheme uses public key cryptography to protect the *integrity* of sessions. During an authentication session, communicating members and the concerned meta servers exchange monitoring messages signed with their private keys. These signed messages, verifiable only with the respective public keys, deliver validating data to help the communicating members ascertain the integrity of the 'encapsulated' run of protocol. It may however be noted that meta authentication exchanges are not *confidential*, ie., exchanged messages are observable by anyone. This is because, meta authentication relies only on the integrity of validation messages, not on their confidentiality. When two members of the same domain authenticate, this message exchange involves those two members and the domain's meta server. When the authentication is between two members belonging to different domains, the exchange involves the two members and the meta servers of both domains. As compared to other public key based systems, the meta authentication scheme imposes only minimal requirements, which are as follows:

1. Each member and meta server in any domain has a public/private key pair.
2. The private key of every member and meta server is kept strictly confidential, known only to the holder of the key (this is a requirement in all public key cryptographic systems).

3. All members in a domain know the public key of the meta server of the same domain.
4. A meta server knows the public keys of all members in its domain.
5. A meta server either knows the public keys of the meta servers in all other domains, or has access to a mechanism through which such keys can be obtained (for example, based on certificates issued by a higher trusted entity).

However,

1. A member need not know the public key of any other member (in its own, or another, domain).
2. A member need not know the public keys of any meta server in other domains.
3. A meta server need not know the public key of any member of other domains.

Thus, the scheme allows each domain to be separately administered. The only inter-domain knowledge is limited to meta servers who need to know the public keys of one another. Thus, compared to schemes where every member needs to know (or is able to access through some mechanism) the public keys of all members in all domains, the overhead is minimal. This also enhances security by reducing the chances of compromised key pairs being used. In large systems, the problem of updating all members when the key pair of a member gets compromised is hard to solve efficiently in a timely manner.

The basic scheme as would be used in *intra-domain* and *inter-domain* meta authentication are shown in Figure 7.1 and Figure 7.2, respectively. In these figures, the public and private keys of any entity X are represented as Kx_pub and Kx_prv, respectively.

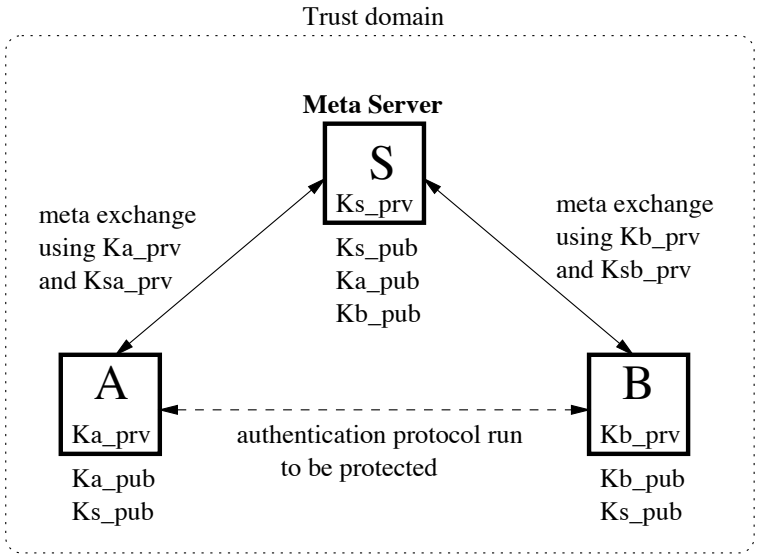


Figure 7.1. *Intra-domain* meta authentication

In the *intra-domain* case shown in Figure 7.1, members A and B belonging to the same *trust domain* engage in an authentication session protected using meta exchanges with the help of meta server S. Every entity (a member or meta server) knows its own public and private keys. In addition, as indicated, the meta server knows the public keys of both members. However, each member needs to know only the public key of the meta server, not that of the other member.

In the *inter-domain* case shown in Figure 7.1, member A of *trust domain 1* and member B of *trust domain 2* engage in authentication protected using meta exchanges with the help of meta servers Sa (*trust domain 1*) and Sb (*trust domain 2*). As indicated, each member needs to know only the public key of the meta server of its own domain, not those of members or meta servers in other domains. Each meta server knows the public keys of the members in its own domain. The figure

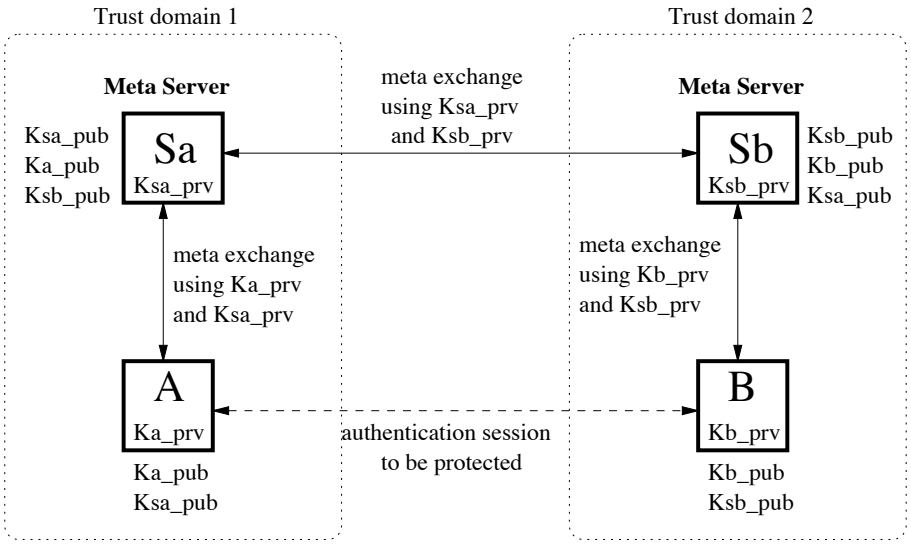


Figure 7.2. *Inter-domain* meta authentication

also indicates that each meta server knows the public keys of meta servers in other domains.

The authentication protocol employed by the communicating members A and B is independent of the meta authentication scheme. Members may use any authentication protocol they choose. It is also irrelevant whether these 'encapsulated' protocols are based on public key or shared key cryptography. Though it is desirable not to use the key-pairs meant for meta authentication (such as Kx_pub and Kx_prv) in the underlying authentication protocols, such use is not precluded.

The following section describes a protocol mechanism, which together with the architecture presented, provides a robust meta authentication service.

7.3. A Protocol for Meta Authentication

As discussed in Section 7.2, the meta authentication framework uses *signed messages* between communicating entities and meta servers to deliver validation data. The reliability of meta authentication depends on the uniqueness of validation data.

During the run of an authentication protocol, several pieces of information are exchanged between the two communicating entities. Such information may include identities, nonces (one-time random values), time-stamps, and shared keys. If the two entities calculate an *integrity check value (ICV)* over these values using a pre-defined and mutually agreed upon function, the results would be identical at the two ends. It is also possible to calculate the *ICV* using weighted functions resulting in different values at the two ends. Still, either member will be able to deduce the value calculated by the other entity, because both of them ideally have access to the same pieces of information. It is desirable to use such weighted functions since it prevents replay type attacks on the meta authentication itself. Other important properties to be satisfied by the chosen function are that, (1) knowing an *ICV* it should be infeasible to generate a set of protocol messages that would result in that *ICV*, ie., the function must be an *irreversible (one-way)* function, and (2) it should be *collision resistant*, meaning, different runs of the protocol must not result in the same *ICV*. Thus, any good hash function may be used for *ICV* generation.

By comparing the values of *ICV* generated at both ends, it can be determined whether the pieces of information seen by the entities are the same or not. By virtue

of encryption mechanisms used in the authentication protocol itself, many of these values will be seen only by the communicating parties and not by any third party. Thus it will be infeasible for an external entity to deduce the *ICV* for a fresh instance of protocol run.

Most attacks against authentication protocols as described in Section 2.4.1 involve manipulation of messages in one form or another, be it replay, alteration, insertion, or deletion. This indicates that in the presence of an attack, one or more pieces of information known to one entity in the session (sent or received) may not be the same as those seen by the entity at the other end. This implies that the *ICV* will not be the same at both ends (or, when weighted functions are used, the value calculated by one will not be same as that deduced by the other).

If there is a reliable and trusted mechanism to exchange and compare these *ICV* values, the entities can determine whether the run of the 'encapsulated' authentication protocol had been tampered with or not. However, there may be cases, depending on the nature of attack, where the external intruding entity may be able to see all the pieces of information seen by legitimate participants. In such cases, simply demonstrating an *ICV* is not sufficient because the intruder also will be able to generate this value. This is the reason why signed messages are used in the meta authentication framework to deliver validation data. The *strong cryptography assumption* ensures that an intruder will not be able to forge the signature of another entity. Thus, the mechanism used in meta authentication involves trusted delivery and comparison of signed integrity check values.

An authentication session protected through meta authentication proceeds through the following steps:

1. Before initiating the authentication session, the authenticating entities agree whether to use the optional meta authentication services. If it is decided not to, then no further protection is available and the entities proceed to *step 3*.
2. If meta authentication is to be used, the entities agree upon a predefined algorithm for weighted integrity check value calculation.
3. The entities run the real authentication protocol. If meta authentication is not used, the entities proceed to *step 14*.
4. As the authentication protocol proceeds, each entity calculates a local *ICV* and 'deduces' the *ICV* that is supposedly being calculated by the other.
5. When the authentication protocol run concludes, each entity sends the 'deduced' *ICV* to the meta server of its domain in a signed (signed with the entity's private key) message that also includes the identity of the entity, identity of the other entity (the other participant in the authentication session) to whom the information is to be delivered, and a time-stamp.
6. Each meta server verifies that the message came from an entity in its own domain, and checks the validity of the signed message received using the public key of the originating entity (this key is known to the meta sever, as the entity is a member of its domain). It also checks the time-stamp to ensure that the message is timely and not a replay.

7. Each meta server then extracts the information and determines whether the recipient entity is a member of its own domain.

   If this is the case (*intra-domain authentication*), it sends the information to the entity in a signed message (signed with the server's private key) also including the identity of this meta server and its own time-stamp. Operation then proceeds to *step 9*.

   However, if the recipient entity is not a member of its own domain (*inter-domain authentication*), then the meta server sends this newly formed message to the meta server of the recipient entity, and operation continues in *step 8*.

8. The receiving meta server verifies the received message for integrity (using the sending meta server's public key) and timeliness. It also verifies that the recipient is a member of its domain. On verification, the server extracts the relevant information (identities of originating and receiving entities, and the 'deduced' ICV value) and sends it to the recipient. The message is signed with this meta server's private key and also includes the meta server's identity and a new time-stamp.

9. On receiving this validation message, each entity verifies that it was delivered by the meta server of its own domain (using the public key of the server, known to all members in the domain), and that it is timely.

10. From the validation message delivered in *step 9*, each entity extracts the identity of the originating entity and verifies that there is indeed an authentication

session proceeding between the two. It then compares the 'deduced' *ICV* value received in the message with the value it had computed locally, and verifies that they are identical. If both these checks are successful, then the entity proceeds to *step 13*.

11. If either of the checks in *step 10* fails, it indicates a possible intrusion attempt and the entity immediately aborts the authentication session. After a timeout, the entity at the other end will notice the absence of response and will also terminate the session. The remaining steps are skipped in this case.
12. If an entity fails to receive the above validation message within a reasonable amount of time after the conclusion of the authentication protocol run, it assumes some foul play and aborts the authentication session. The other entity will also have to terminate the session after a timeout. The remaining steps are skipped in this case.
13. The successful checks in *step 10* assure both authenticating entities that the execution of the authentication protocol itself has been proper and secure.
14. Based on the outcome of the real authentication protocol, the entities determine whether to accept the authenticity of the other entity or abort the session.

The procedure described above in steps *1* through *14* constitute the *meta authentication protocol*. As described, it is applicable to both *intra-domain* and *inter-domain* authentication scenarios.

The high level message flows for these two cases are shown in Figure 7.3 and Figure 7.4, respectively.

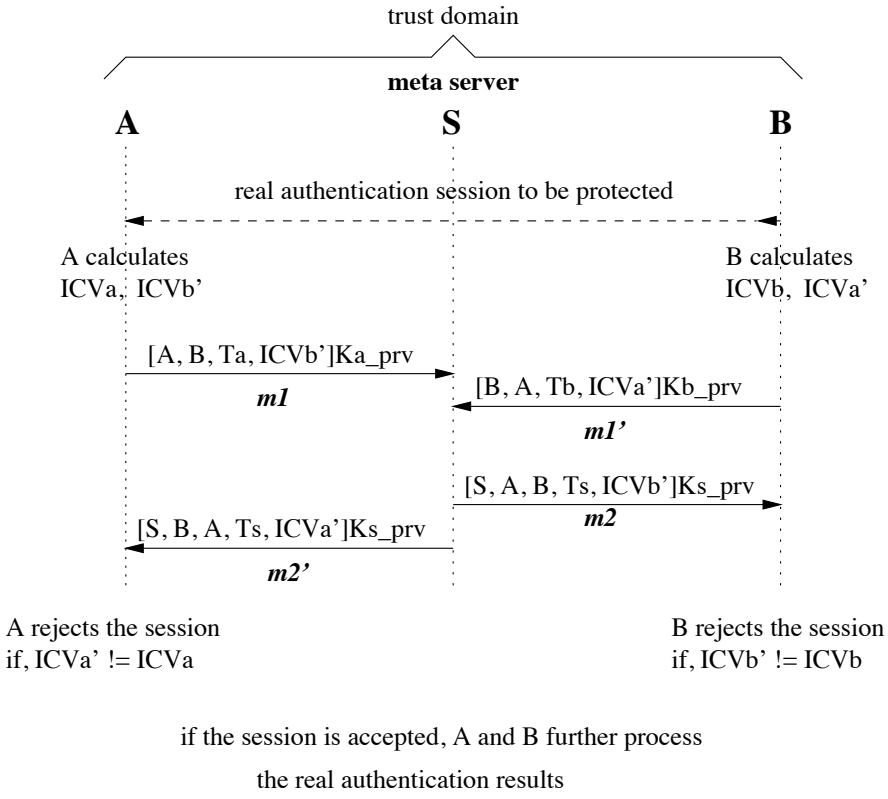


Figure 7.3. *Intra-domain* meta authentication protocol

In Figure 7.3, $m1, m2$ show the path of validation message delivery from A to B, and $m1', m2'$ show the path from B to A. Note that there is no timing relationship between the two sequences, timing is applicable only within the same sequence. Similarly in Figure 7.4, $m1, m2, m3$ and $m1', m2', m3'$ show the delivery path. A message of the form $[p, q, r, ...]Kp\_prv$ in fact represents the message $\{p, q, r, ..., (p, q, r, ...)Kp\_prv\}$, where, $(p, q, r, ...)Kp\_prv$ is the originator's signature; ie., each message is a concatenation of some clear-text and the corresponding validating signature.

Later sections demonstrate how this protocol prevents various attacks on authentication protocols. However, before proceeding to demonstrate its use, a more

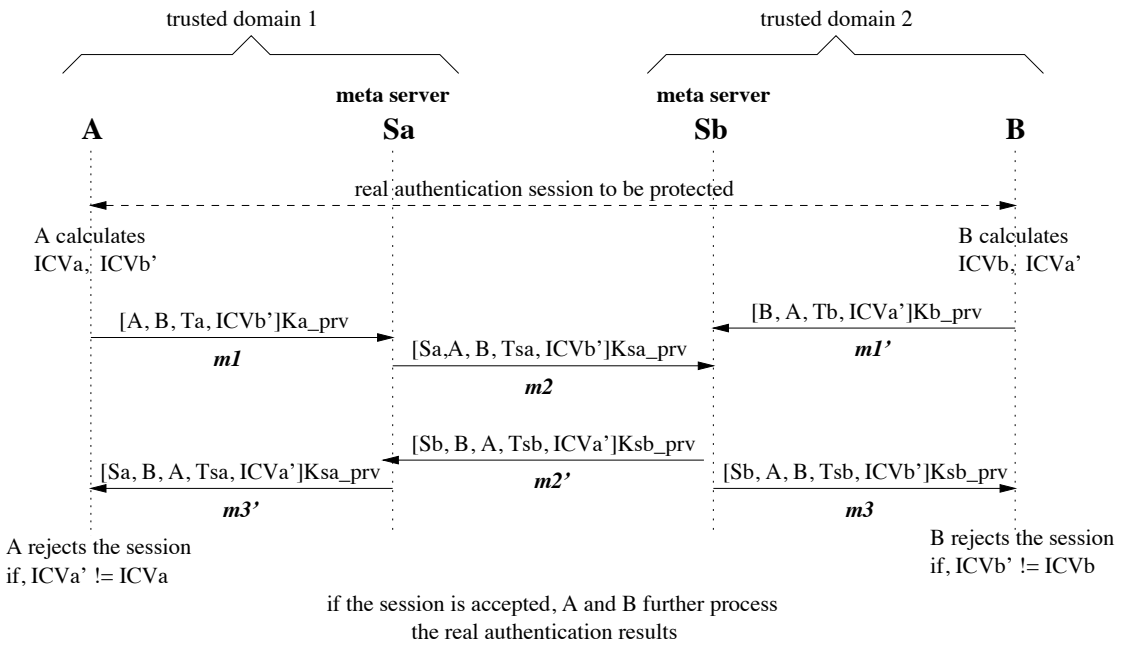


Figure 7.4. *Inter-domain* meta authentication protocol

formal argument on the security properties of the meta authentication protocol is given in the following section.

7.3.1. Security of Meta Authentication Protocol

A qualitative explanation of meta authentication was given in preceding sections. A more formal argument on the security of meta authentication is given here.

Consider the high level view of the transfer of validating information in the meta authentication model shown in Figure 7.5, and the trusted paths shown in Figure 7.6.

In these figures, x and y are the *integrity check values (ICV)* calculated locally by A and B. Also, y' is the value deduced by A (corresponding to y calculated by B), and x' is the value deduced by B (corresponding to x calculated by A).

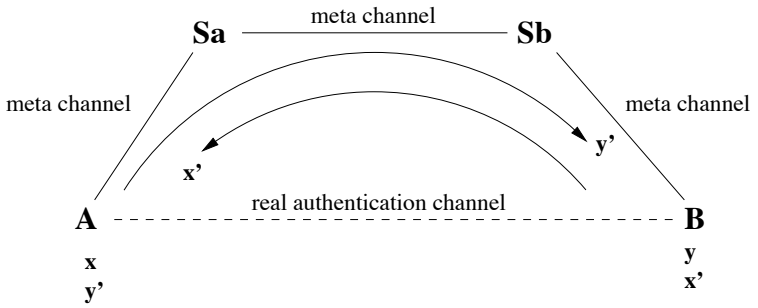


Figure 7.5. Validation data transfer in meta authentication

The security of the meta authentication scheme can be demonstrated through the following argument:

1. Entity A accepts the authentication protocol run if and only if $x' = x$.
   Entity B accepts the authentication protocol run if and only if $y' = y$.
2. Given x or y, it is infeasible to generate a set of messages that give the same *ICVs*. This is because of the *collision resistance* and *irreversibility* of the *ICV* generating function.
3. A and B exchange y' and x' through a sequence of signed messages as indicated in Figure 7.6.

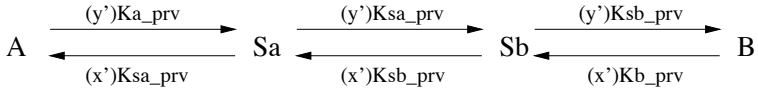


Figure 7.6. Trusted paths in meta authentication

4. Under the given assumptions and based on the properties of public key cryptography, a malicious entity cannot do the following:

(a) *forge* any of the signed messages in (3), because the required private key will not be known to it. This follows from the *strong cryptography* assumption.

(b) *replay* any of the messages in (3), because x and y are time-dependent since their calculation uses time-stamps as one of the inputs.

(c) *substitute* a message in (3) with a similar message from a different run of the protocol, because (2) ensures that it is infeasible to manipulate a protocol run such that it results in a known ICV (ie., x or y).

Therefore, if a message is delivered in (3), the receiver is assured of the *data origin authenticity* and *data integrity* (note that *data confidentiality* is not assured, but meta authentication does not rely on confidentiality). Any attempt to carry out (a), (b), or (c) will be detected by the receiver, which then promptly aborts the session.

However, a malicious entity capable of observing and manipulating protocol messages may be able to do the following:

(a) *intercept* and possibly delete a message in (3). But the receiver will detect the message loss through timeout and abort the session.

Therefore, if a message is not delivered in (3), the receiver will terminate safely and stop all further processing of the 'encapsulated' protocol.

Thus tampered or manipulated messages and lost messages result in the termination of the authentication session, ie., the meta protocol is *fail safe*.

5. From (4), it follows that the meta channel between A and B is *secure* (in the limited sense of providing integrity and authenticity without any confidentiality) and *fail safe*. This implies that, any message that is successfully delivered and accepted is *valid, fresh,* and *authentic*.
6. From (5), it follows that:
   (a) if an *ICV* match is found, then the *integrity* of the monitored authentication protocol is assured.
   (b) if an *ICV* match is not found, then the monitored protocol run did not proceed properly.
   (c) failure of the meta protocol itself (through loss of messages, for example) leads to safe termination, and is equivalent to (b).
7. From (6), it follows that an 'encapsulated' (monitored) authentication protocol is secure (in terms of *integrity*) if the meta channel between the authenticating entities is secure and fail safe.

Thus, the meta authentication scheme provides a robust mechanism to ensure the integrity of authentication protocols.

The following section demonstrates how meta authentication scheme can be used to protect against various attacks on authentication protocols.

7.4. Protecting Authentication Protocols Against Attacks

Authentication protocols are vulnerable to a variety of attacks as discussed in Section 2.4.1. Most of these attacks exploit vulnerabilities in protocol structure and belief establishment mechanisms, and do not depend on the strength (or lack thereof) of any specific encryption mechanism used in the protocol. *Replay attacks, oracle session attacks,* and *parallel session attacks* are examples of potential threats to authentication protocols. This section demonstrates how such attacks may be defended against using the meta authentication scheme, by considering a variety of known attacks against different types of protocols. The more general scenario of *inter-domain* authentication is assumed in all these cases, since *inter-domain* authentication can be considered as a subset of the former.

7.4.1. Preventing Parallel Session Attacks

This section shows, using the example of a simple *shared key one-way authentication* protocol, how meta authentication can prevent *parallel session attacks*. An entity A tries to authenticate another entity B using the shared key protocol; B does not try to authenticate A in this case, so the authentication is one-way. This simple protocol is vulnerable to a *parallel session* attack wherein an intruder C manages to falsely convince A that it is in fact B. It does this by engaging in two parallel sessions with A. B need not even be involved in any session while the attack occurs.

The authentication protocol and the parallel session attack are shown in Figure 7.7 and Figure 7.8, respectively.

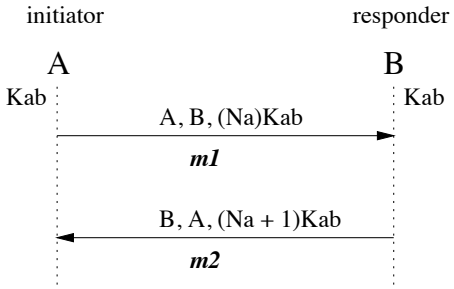


Figure 7.7. Simple one-way authentication protocol

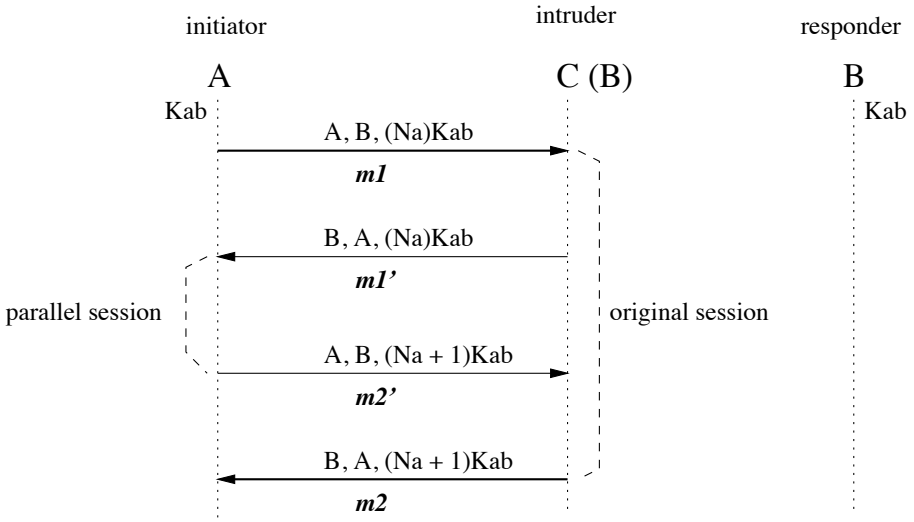


Figure 7.8. Parallel session attack on simple one-way protocol

In the normal protocol run (Figure 7.7), A sends a nonce Na to B (message $m1$), encrypted using their shared key Kab. If B is able to send back $(Na + 1)$ (message $m2$) encrypted with the same shared key, A assumes that it is indeed communicating with B. In the attack (Figure 7.8), a malicious entity C intercepts message $m1$ from

A. As it does not have access to Kab, C cannot recover Na. But, using the same encrypted value, C starts another parallel session with A by sending message $m1'$. Believing that B has initiated a new session, A sends back the encrypted value of $(Na+1)$ in message $m2'$. Though C cannot decrypt this value either, it can use the encrypted value as it is to form the response expected in the original session initiated by A (as this is exactly the value A expects to receive from 'B'). When the message $m2$ generated by C arrives, A concludes that it must have come from B. Thus, C succeeds in masquerading as B.

Figure 7.9 shows the scenario when meta authentication is employed.

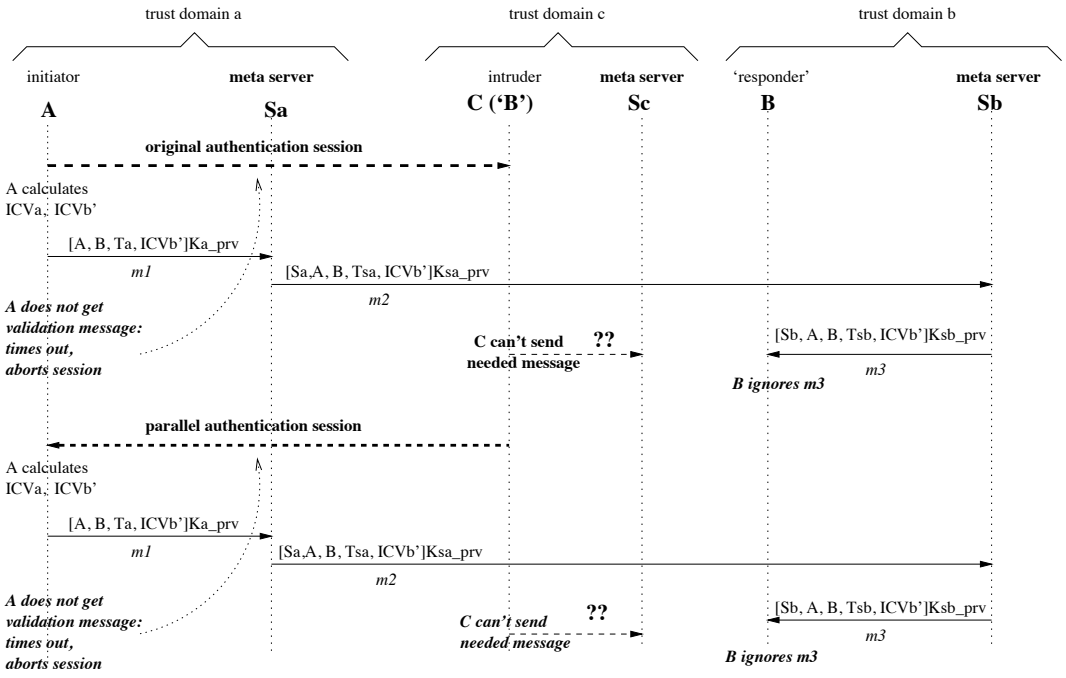


Figure 7.9. Preventing *parallel session attack* using meta authentication

As per the meta authentication protocol rules, A calculates $ICVa$ and $ICVb'$ (assuming that it is talking to B) and sends the validation message containing $ICVb'$ to B (via Sa and Sb). However, when B receives this message, it recognizes that it has currently no session with A (because A's initiation message was intercepted by C and never reached B). Hence, the message will be simply discarded by B. It may be noted that, even if C manages to intercept the validation message itself as well, no harm is done as B would have discarded it anyway.

Meanwhile, A is expecting the corresponding message from B. Since B is not a participant in the session, it will not send any message. Even if it wants, C will not be able to send a fake validation message to A. To do that, C should be able to calculate the ICV and be able to get Sc to deliver a signed message on its behalf to A (via Sa). But, without Kab, C cannot see the information carried inside the messages of the real authentication protocol, and hence will not be able to deduce the ICV. Also, to be accepted by A, the message must have B as the identity of its originator. Because of this, C cannot use its own identity in the message, as it would be rejected by A who is not expecting anything from C.

If C tries to use B's identity in the message sent to Sc (to be forwarded to A via Sa), Sc will detect the discrepancy between the claimed identity ('B') and the signature, and refuse to deliver it. A possible conspiracy between C and Sc will also not work because, if Sc forwards the 'validation' message mentioning B as the originating entity, it would be rejected by Sa when it notices that the message is signed by Sc which is not a meta server for B's domain.

In short, C will not be able to deliver an acceptable validation message to A, either during the original session initiated by A or during the parallel session started by C. Therefore, according to the meta authentication rules, A will time out and abort the ongoing authentication sessions for which the decisions of acceptance are pending. Thus, C's attempt to pass off as B through the parallel session attack is defeated.

7.4.2. Preventing Replay Attacks

In the Needham-Schroeder shared key protocol discussed in Section 2.3.2.2, two trusted entities, A and B, use the services of a trusted authentication server, AS, to establish fresh session keys to be used in their communication. There is a known *freshness attack* (*replay attack*) against this protocol as described in Section 2.4.2.1. The original protocol and the attack are reproduced for convenience in Figure 7.10 and Figure 7.11, respectively. In the attack, a malicious entity C replays a message from an old session between A and B, and succeeds in falsely convincing B that it is indeed A. The trusted authentication server is not even involved during the session.

This attack can be prevented by executing the protocol under the meta authentication scheme. Figure 7.12 shows the scenario.

As shown in Figure 7.12, the unsuspecting responder B calculates and sends the ICV to A (the claimed initiator) via Sb and Sa, and then waits for a similar message to arrive from 'A' in order to make a decision on whether to proceed with the session.

However, A ignores the validation message from B since it is not expecting any such message from B with whom it does not have any session in progress. Due to the same reasons as given in Section 7.4.1 for the case of the simple shared key protocol, the intruder C will not be able to get an acceptable validation message delivered to B. Therefore, after waiting, B will eventually time out and abort the real authentication session for which the decision for acceptance is pending. Thus C's attempt to fake A's identity through the replay (freshness) attack fails.

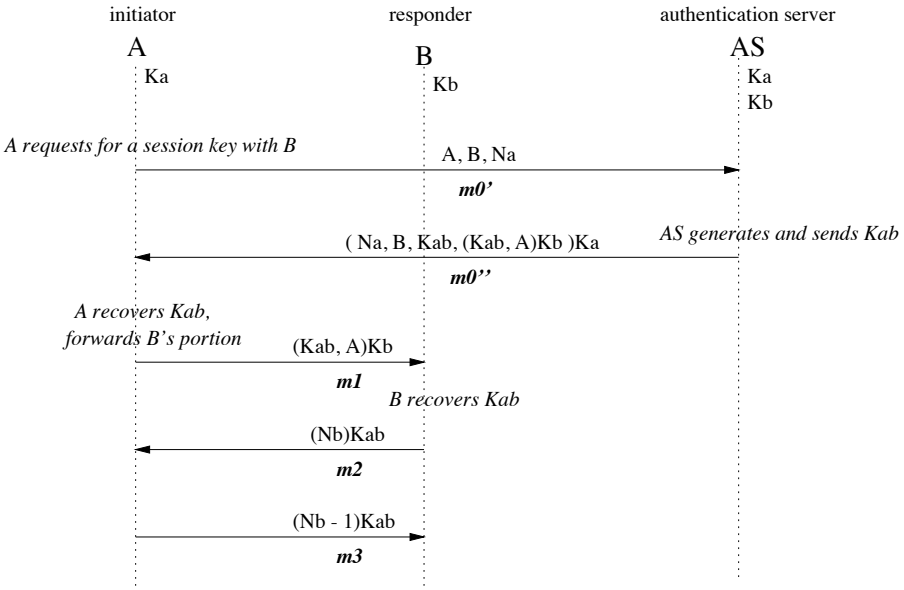


Figure 7.10. Needham-Schroeder shared key protocol

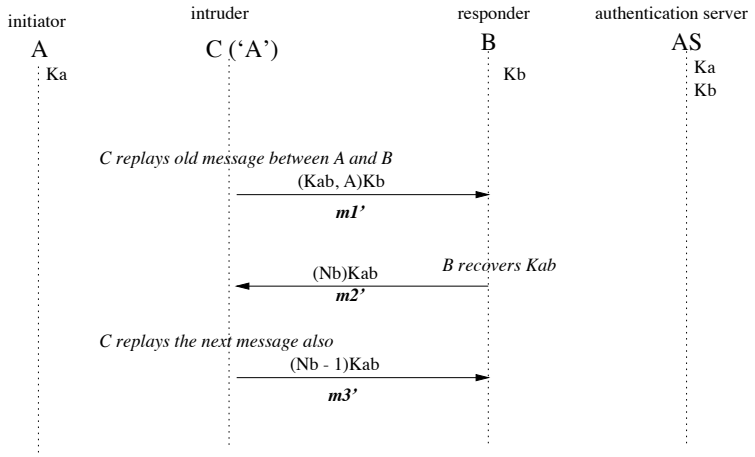


Figure 7.11. Freshness attack on Needham-Schroeder shared key protocol

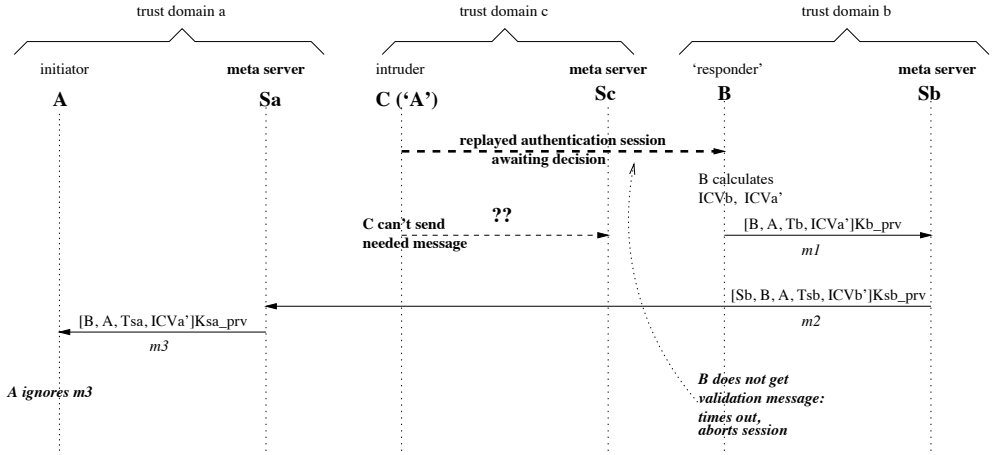


Figure 7.12. Preventing *replay (freshness) attack* using meta authentication

7.4.3. Preventing Oracle Session Attacks

In the Needham-Schroeder public key authentication protocol, described in Section 2.3.2.3, two trusted entities, A and B, use the knowledge of their respective private keys to prove identity. As explained in Section 2.4.2.2, this protocol is vulnerable to an *oracle attack*. A malicious third entity, C, who also happens to be a legitimate member of the domain, waits until A initiates an authentication session with it. C then initiates a fake session with B, using A as an 'oracle' to generate messages needed in the fake session. Through the clever manipulation of messages, C succeeds in faking the identity of trusted entity A. A simplified version of the original protocol and the attack scenario are shown in Figure 7.13 and Figure 7.14, respectively.

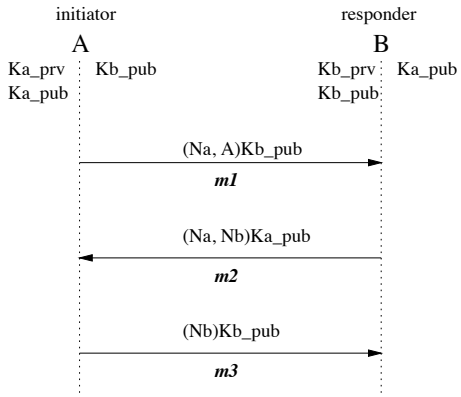


Figure 7.13. Needham-Schroeder public key protocol

By subjecting the protocol to the supervision of the meta authentication scheme, the attack can be prevented from succeeding. Figure 7.15 shows the scenario.

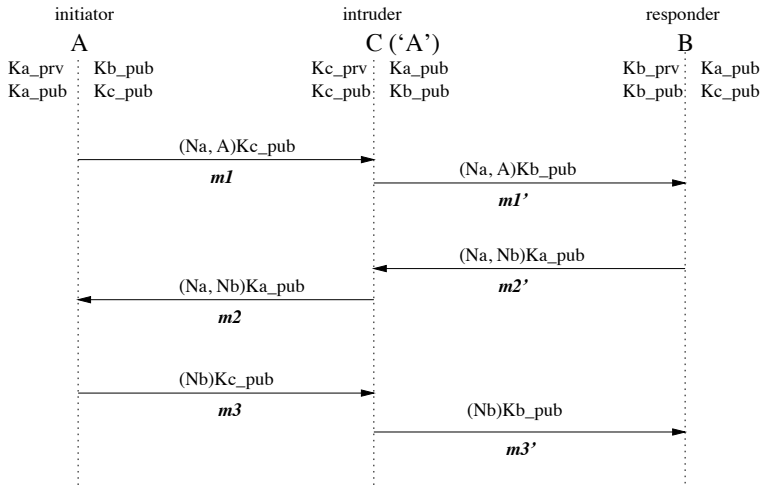


Figure 7.14. Oracle attack on Needham-Schroeder public key protocol

As shown in Figure 7.15, there are two sessions to be considered. The first is the legitimate session initiated from A to C. The other is the fake session initiated from C (in the guise of A) to B. In the first session, both A and C are able to exchange validation messages with each other (m1, m2, m3, and m1', m2', m3') and therefore the session is accepted as properly executed.

In the fake session from C to B, B calculates and sends the ICV to A (the claimed initiator) via Sb and Sa (messages m1", m2", m3"), and waits for a message from 'A'. However, A ignores this message as it is not expecting any message from B. Again, for the same reasons as explained in Section 7.4.1, the intruder C will not be able to deliver an acceptable validation message to B through Sc and Sb. Thus, B will eventually time out and abort the authentication session to 'A' (actually C) for which acceptance decision is pending. This defeats C's attempt to take on A's identity through the oracle session attack.

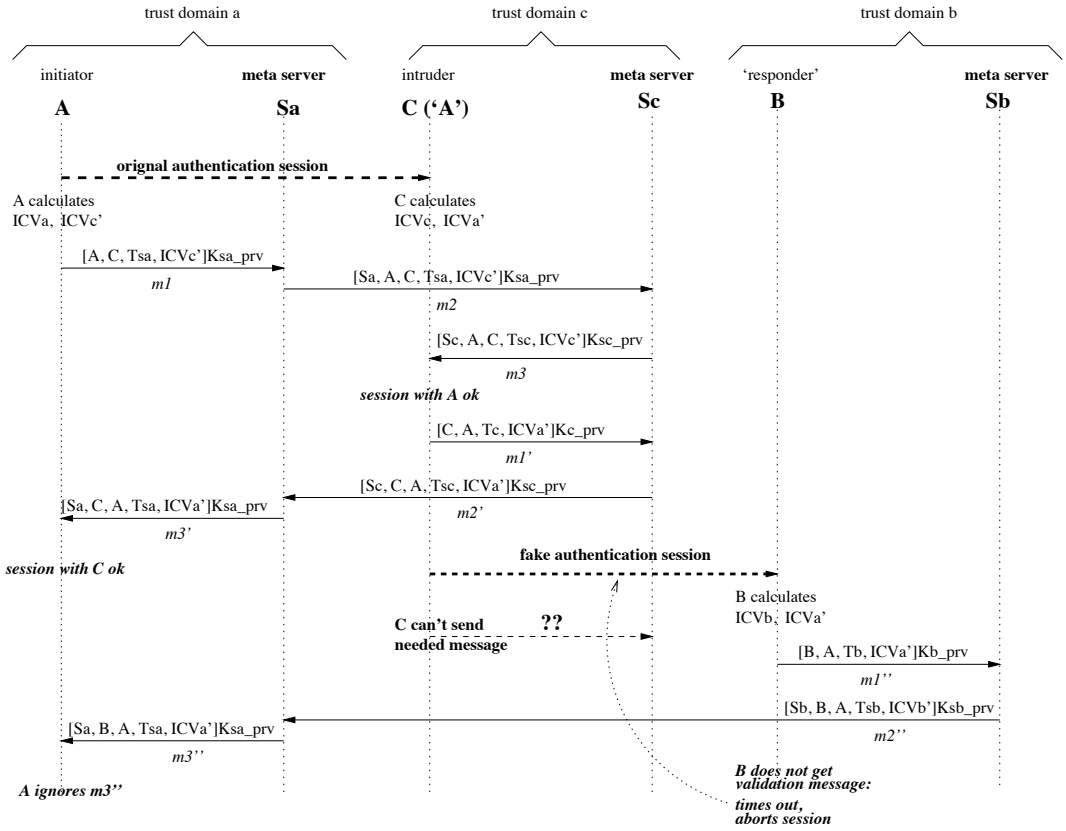


Figure 7.15. Preventing *oracle session attack* using meta authentication

7.4.4. Preventing Binding Attacks

Binding attacks are generally not aimed directly at authentication protocols, but is equally dangerous. The attack compromises the security and trustworthiness of *directory servers* commonly used in public key encryption infrastructures to distribute public keys of entities. Entity A, that wants to send encrypted messages to another entity B, requests a directory server D for B's public key, by sending a message containing the identity of B and a nonce. The server D sends the requested key to A, along with its own identity, the requester's identity, the nonce, and the requested

public key. This reply message is signed with D's public key which is considered to be well known. A may then proceed to send encrypted messages to B using the public key of B thus obtained.

In the attack, an intruder C intercepts the original request from A and then forwards it to D after replacing the identity of B with its own identity (C). The reply from D, which contains the public key of C, is forwarded to A by C. The unsuspecting A starts sending encrypted messages to B using the public key received in the response. As the returned key is that of C and not that of B, C can intercept and decrypt all the messages. Even if B happens to see the messages, it will not be able to decrypt them. The attack is called a *binding attack* because it compromises the 'binding' between an entity and its public key.

The original protocol and the attack scenario are shown Figure 7.16 and Figure 7.17, respectively.

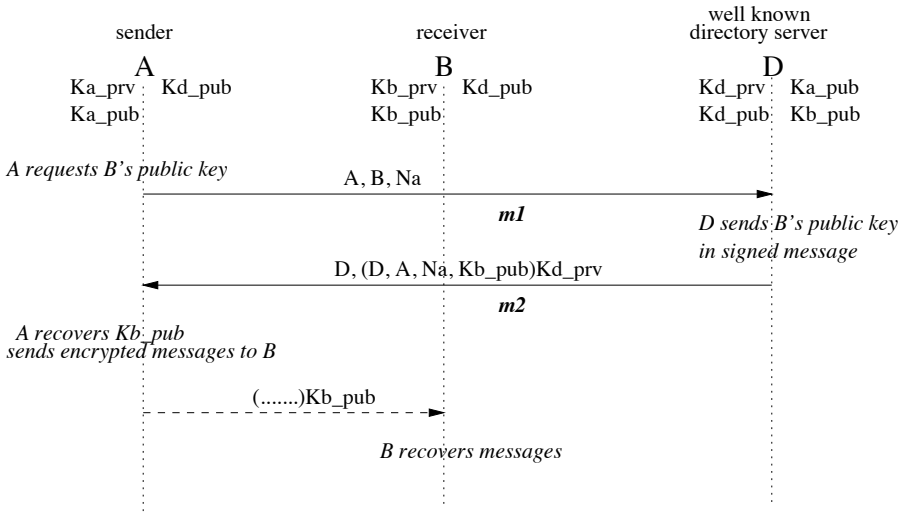


Figure 7.16. Public key distribution protocol

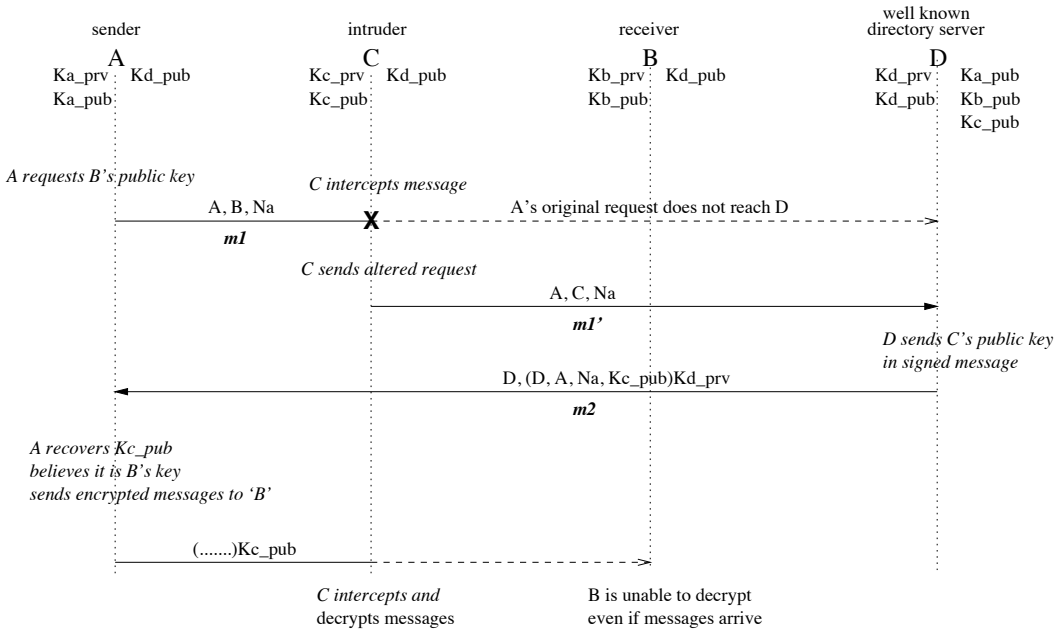


Figure 7.17. Binding attack on public key distribution protocol

The meta authentication scheme can be used in this case also to defend against the attack. Though the protocol being protected is not an authentication protocol, this example illustrated the more general applicability of the meta authentication scheme. Figure 7.18 shows how the scheme handles the attack.

In this case, since the messages seen by A and D will be different (because of the alterations by C), the ICV calculated by them will be different. Thus when the validation message from D arrives, A detects the discrepancy and discards the received public key 'Kb_pub' (in reality, Kc_pub). Since this key will not be used by A for encrypting any messages, C's intention of snooping on messages meant for B will not succeed.

It is evident that a wide variety of attacks against authentication protocols (and other security protocols such as the key binding protocol in the last example) can be

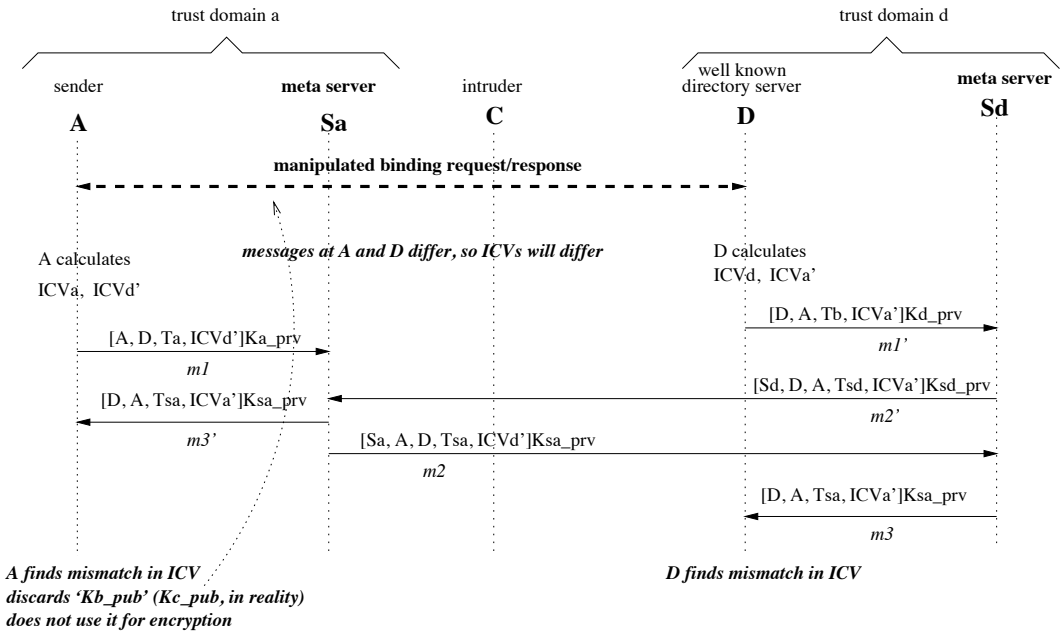


Figure 7.18. Preventing *binding attack* using meta authentication

prevented by employing the meta authentication scheme. Further, as this example shows, the meta authentication scheme has more general applicability than being just limited to protecting authentication protocols.

Thus, the meta authentication paradigm appears to hold much promise in defending networks against various forms of malicious activities.

Chapter 8

CONCLUSION AND FUTURE DIRECTION

This chapter provides a summary of the work presented in this thesis. It also presents an outline of potential future directions for further research in the areas investigated in the thesis.

8.1. Thesis Summary

This thesis is the result of an investigation into the potential vulnerabilities of authentication systems, attempts to detect such weaknesses, and techniques to defend against the exploitation of those weaknesses.

Chapter 2 discussed the principles of operation of authentication protocols and some of their potential weaknesses, along with examples of attacks that exploit such weaknesses to breach security. The need for formal analysis and verification techniques in the case of authentication protocols was discussed and a survey of previous approaches presented in Chapter 3.

A verification framework for authentication systems was developed, based on state machine principles, in Chapter 4. Authentication scenario modeling, and a demonstration of the use of the framework in verification was presented in Chapter 5.

An alternative approach, based on simulation, was developed in Chapter 6. The approach, which stresses validation rather than verification, was shown to be potentially useful in identifying protocol flaws. Though not as rigorous as formal techniques, this simulation approach promises to be a valuable and powerful tool in studying authentication protocols, especially considering its relative simplicity and flexibility as compared to formal verification techniques.

After investigating these different methods to model and analyze authentication systems, and identify their weaknesses, a *meta authentication* scheme to protect authentication even in the presence of vulnerabilities was developed in Chapter 7. An architecture and protocol for meta authentication were developed, and their effectiveness demonstrated by showing how various attacks can be prevented.

8.2. Future Directions

In each of the areas mentioned in the previous section, there is ample opportunity for further research. The following sections briefly describe some potential future directions in these areas.

8.2.1. Verification Techniques

The state machine based verification framework developed in Chapter 4 is a potentially powerful tool to analyze authentication systems. However, the verification procedure is still complex and require fairly good expertise in protocol modeling

and representation. Therefore it is felt that an automated system with a simplified protocol modeling interface may prove valuable. This would be an area that needs further research and could provide very useful results.

Devising a simple and flexible language for expressing authentication protocols and properties would be another useful contribution that needs to be investigated further. Integration of such a language into the verification framework could yield a very flexible and powerful verification framework that would be useful for researchers and protocol designers alike.

8.2.2. Simulation Based Validation

As demonstrated in Chapter 6, discrete event simulation appears to be a promising approach to validate authentication protocols. In the work presented in that chapter, modeling of the protocol and its translation into a simulation framework was performed manually. It would be a very worthwhile effort to investigate the feasibility of automating this process. Such a modified setup would provide a valuable environment to study authentication protocols.

Another area to be investigated is regarding the coverage provided by simulation, and hence its effectiveness as a dependable validation tool.

8.2.3. Meta Authentication

Chapter 7 developed the *meta authentication scheme* as an effective way to counter various types of attacks against authentication protocols. As demonstrated, the results appear to be promising and consistent. One major assumption behind the scheme's operation was that there would be trust domains based on public key cryptography. In practice, this may prove to be somewhat of a restriction, especially with the emergence of small networked devices that may not have enough computational resources to carry out expensive public key transformations.

One very promising area to investigate is, whether it is possible to remove the above mentioned restriction without jeopardizing the security and robustness of the scheme. Though it seems unlikely that it would be possible to totally do away with encryption operations, there is hope that techniques developed for agreement and consensus in distributed systems may prove valuable in devising techniques to provide an efficient solution.

With the emergence of *IPsec* as a standardized framework for Internet security [3] [21], it is important to investigate how a scheme like meta authentication can coexist with, and contribute to it. This is especially true in environments where minimal overhead (in terms of computational resources, communication bandwidth, and managability) is important as in the case of increasingly popular mobile networks with low power devices. This is another potential area for further research.

8.3. Conclusion

The design of authentication protocols is both exciting and challenging. It is also notoriously full of potential pitfalls. Even a very slight lack of attention to details could result in protocols with subtle vulnerabilities that are hard to detect and defend against. Security of entire networks could be jeopardized by such protocols, with very serious and far reaching implications.

This thesis investigated potential vulnerabilities in authentication protocols, to develop methods to detect such weaknesses, and to devise techniques to defend against the exploitation of those weaknesses that might still escape careful design and thorough analysis, and creep into protocols deployed in real data networks.

Evidently, there is considerable opportunity for further research into each of the discussed areas. It is hoped that the work presented in this thesis may trigger further investigations in this very rewarding and important field - the art and science of authentication protocols.

REFERENCES

[1] Abadi, M., and Needham, R. Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering 22*, 1 (Jan. 1996), 6–15.

[2] Amoroso, E. *Fundamentals of Computer Security Technology.* Prentice-Hall, Englewood Cliffs, NJ, 1994.

[3] Atkinson, R., and Kent, S. *Security Architecture for the Internet Protocol.* The Internet Society, Nov. 1998. RFC 2401.

[4] Aura, T. Modelling the needham-schroeder athentication protocol with high level petri nets. Digital Systems Laboratory Report B14, Helsinki Univ. of Technology, Sep. 1995. (http://saturn.hut.fi/html/staff/tuomas.html).

[5] Baker, R. H. *Computer Security Handbook.* McGraw-Hill, NY, 1991.

[6] Bartlett, K. A., Scantlebury, R. A., and Wilkinson, P. T. A note on reliable full-duplex transmission over half-duplex links. *Communications of the ACM*, 12 (1969).

[7] Bird, R., Gopal, I., Herzberg, A., Janson, P. A., Kutten, S., Molva, R., and Young, M. Systematic design of a family of attack-resistant authentication protocols. *IEEE Journal on Selected Areas in Communications 11*, 5 (June 1993), 679–693.

[8] Bochmann, G. V. Finite state description of communication protocols. *Computer Networks*, 2 (1978), 361–372.

[9] Brand, D., and Zafiropulo, P. On communicating finite-state machines. *Journal of the ACM 30*, 2 (Apr. 1983), 323–342.

[10] Burrows, M., Abadi, M., and Needham, R. A logic of authentication. Research Report SRC-39, DEC Systems Research Center, Palo Alto, CA, 1989.

[11] Burrows, M., Abadi, M., and Needham, R. A logic of authentication. *ACM Transactions on Computer Systems 8*, 1 (Feb. 1990), 18–36.

[12] Clark, J., and Jacob, J. A survey of authentication protocol literature. Survey report, University of York, UK, 1996. (http://www.cs.york.ac.uk/ jac/).

[13] Danthine, A., and Bremer, J. Modelling and verification of end-to-end transport protocols. *Computer Networks*, 2 (1978), 381–395.

[14] Davies, D. W., and Price, W. L. *Security for Computer Networks.* John Wiley & Sons, Ltd., Chichester, UK, 1984.

[15] Denning, D. E., and Sacco, G. M. Time stamps in key distribution protocols. *Communications of the ACM 24*, 8 (Aug. 1981), 533–536.

[16] Devargas, M. *Network Security.* NCC Blackwell, Oxford, UK, 1993.

[17] Dolev, D., and Yao, A. C. On the security of public key protocols. *IEEE Transactions on Information Theory IT-29*, 2 (Mar. 1983), 198–208.

[18] Gollmann, D. What do we mean by entity authentication? In *Proceedings of the IEEE Symposium on Security and Privacy* (1996), IEEE Computer Society Press, 46–54.

[19] Gong, L., Needham, R., and Yahalom, R. Reasoning about belief in cryptographic protocols. In *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy* (1990), IEEE Computer Society Press, 234–248.

[20] Gong, L., and Syverson, P. Fail-stop protocols: An approach to designing secure protocols. In *5th International Working Conference on Dependable Computing for Critical Applications (DCCA-5)* (1995), Springer-Verlag, Heidelberg, Germany, 44–45.

[21] Harkins, D., and Carrel, D. *The Internet Key Exchange (IKE).* The Internet Society, Nov. 1998. RFC 2409.

[22] Holzmann, G. J. *Design and Validation of Computer Protocols.* Prentice-Hall, Englewood Cliffs, NJ, 1991.

[23] Hopcroft, J. E., and Ullman, J. D. *Introduction to Automata Theory, Langauges, and Computation.* Addison-Wesley Publishing Company Inc., Reading, MA, 1979.

[24] Hoskote, Y. V. *Formal Techniques for Verification of Synchronous Sequential Circuits.* PhD thesis, University of Texas, Austin, 1995.

[25] Hoskote, Y. V., Abraham, J. A., and Fussell, D. S. Automated verification of temporal properties specified as state machines in vhdl. In *Proceedings Fifth Great Lakes Symposium on VLSI,* Buffalo, NY (Mar. 1995), 100–105.

[26] Indiradevi, K., and Nair, V. S. S. Usability and reliability evaluation of user interfaces. *Proceedings of IEEE International Workshop on Evaluation Techniques for Dependable Systems,* San Antonio, TX (Oct. 1995).

[27] Kailar, R., and Gligor, V. D. On belief evolution in authentication protocols. In *Proceedings of the IEEE Computer Security Foundations Workshop,* Franconia, NH (1991). (http://www.bnetal.com/raja/).

[28] Katzela, I. *Modeling and Simulating Communication Networks: A Hands-On Approach Using OPNET.* Prentice-Hall, Englewood Cliffs, NJ, 1999.

[29] Lowe, G. An attack on the needham-schroeder public key authentication protocol. *Information Processing Letters* (Nov. 1995), 131–136.

[30] Mano, M. M. *Digital Design, 2 ed.* Prentice-Hall, Englewood Cliffs, NJ, 1991.

[31] Meadows, C. A. The nrl protocol analyzer: An overview. In *Proceedings of the 2nd Conference on the Practical Applications of Prolog* (1994), Association for Logic Programming.

[32] Meadows, C. A. Formal verification of cryptographic protocols: A survey. In *Advances in Cryptology - Asiacrypt '94, LNCS 917* (1995), Springer-Verlag, Heidelberg, Germany, 133–150.

[33] Michel, J., Balaban, P., and Sharunugan, K. S. *Simulation of Communications Systems, 2 ed.* Kluwer Publishers, USA, 2000.

[34] Mil3. *Opnet Documentation.* MIL 3, Inc., Washington DC, USA, 1997. (http://www.mil3.com).

[35] Millen, J. K. The interrogator model. In *Proceedings of the IEEE Symposium on Security and Privacy* (1995), IEEE Computer Society Press, 251–260.

[36] Millen, J. K., Clark, S. C., and Freedman, S. B. The interrogator: protocol security analysis. *IEEE Transactions on Software Engineering SE-13,* 2 (Feb. 1987), 274–288.

[37] Mitchell, J. C., Mitchell, M., and Stern, U. Automated analysis of cryptographic protocols using mur-phi. In *Proceedings of the IEEE Symposium on Security and Privacy* (1997), IEEE Computer Society Press, 141–151.

[38] Nair, V. S. S., Abraham, J. A., and Indiradevi, K. Formal checking of reliable user interfaces. *First Conference on Fault-Tolerant Systems,* Madras, India (Dec. 1995).

[39] Needham, R. M., and Schroeder, M. D. Using encryption for authentication in large networks of computers. *Communications of the ACM 21,* 12 (Dec. 1978), 993–999.

[40] Nelson, V. P., Nagle, H. T., Carroll, B. D., and Irwin, J. D. *Digital Logic Circuit Analysis & Design.* Prentice-Hall, Englewood Cliffs, NJ, 1995.

[41] Neuman, B. C., and Ts'o, T. Kerberos: An authentication service for computer networks. *IEEE Communications 32*, 9 (Sep. 1994), 33–38.

[42] Oppliger, R. *Authentication Systems for Secure Networks.* Artech House, Inc., Norwood, MA, 1996.

[43] Postel, J. B. *A Graph-Model Analysis of Computer Communication Protocols.* PhD thesis, University of California, Los Angeles, 1974.

[44] Rivest, R. L., Shamir, A., and Adleman, L. A method for obtaining signatures and public-key cryptosystems. *Communications of the ACM 21*, 2 (Feb. 1978), 120–126.

[45] Sadiku, M., and Ilyas, M. *Simulation of Local Area Networks.* CRC Press, Boca Raton, FL, 1995.

[46] Sauer, C. *Simulation of Computer Communication Systems.* Prentice-Hall, Englewood Cliffs, NJ, 1983.

[47] Schneider, S. Security properties and csp. In *Proceedings of the IEEE Symposium on Security and Privacy* (1996), IEEE Computer Society Press, 174–187.

[48] Schneier, B. *Applied Cryptography - Protocols, Algorithms, and Source Code in C, 2 ed.* John Wiley & Sons, Inc., NY, 1996.

[49] Syverson, P. F., and van Oorschot, P. C. On unifying some cryptographic protocol logics. In *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy* (1994), IEEE Computer Society Press, 14–28.

[50] Zafiropulo, P., West, C. H., Rudin, H., Cowan, D. D., and Brand, D. Protocol analysis and synthesis using a state transition model. In *Computer Network Architectures and Protocols*, J. P. E. Green, ed., Plenum Press, NY, 1982, 645–669.