ON THE MODELING, DESIGN, AND IMPLEMENTATION OF FIREWALL TECHNOLOGY

A Thesis

Submitted to the Faculty

of

Purdue University

by

Christoph Ludwig Schuba

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

December 1997

Meinen Eltern

ACKNOWLEDGMENTS

I thank all those individuals and friends who have contributed, both professionally and personally, something valuable and irreplaceable in my path through graduate school at Purdue University and to the completion of my doctoral studies.

I thank my major professor, Dr. Eugene H. Spafford, and my advisor at Xerox Palo Alto Research Center (PARC), Dr. J. Bryan Lyles, for their time, for sharing their profound professional insights, and for their friendship. I appreciate the contributions of the other committee members, Dr. Douglas E. Comer and Dr. J. Timothy Korb.

My thanks to the administration, faculty, staff, and students of the Department of Computer Sciences at Purdue University and the administrative, research, and support staff at Xerox PARC. I especially thank the members of the Computer Operations, Audit and Security Technology (COAST) Laboratory. It was one of the perquisites of my graduate studies to have been part from the very beginning of this diverse, challenging, and successful group of people organized around Dr. Spafford's vision of a laboratory dedicated to practical computer security research.

Several people deserve special mention: Dr. William J. Gorman III and Mary Jo Maslin in the graduate student office for making red tape transparent; Marlene Walls for administrative support; Dr. Michal Young and Katherine Price for valuable technical discussions; Mary-j Klang, Dr. Sandeep Kumar, Karthic Nataraj, and Thomas Tarman for reviewing early drafts of this dissertation; Claudia Fajardo-Lira, Carlos Gonzalez Ochoa Aleman, Mary Jo Maslin, and Dr. Eugene H. Spafford for being my friends when it really counted.

Very special thanks to Mary-j Klang for her love, friendship, and support which I experienced in so many different ways. And last, but not least my heartfelt thanks to my family for their love and encouragement.

My studies and research were funded in part by Sprint Corporation, Xerox Corporation, Sun Microsystems, and the bi-national German-American Fulbright Commission. I gratefully acknowledge their financial support.

TABLE OF CONTENTS

Page

Page

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

ABSTRACT

Schuba, Christoph Ludwig. Ph.D., Purdue University, December 1997. On the Modeling, Design, and Implementation of Firewall Technology. Major Professor: Eugene H. Spafford.

This dissertation studies one particular aspect of providing communication security: firewall technology. Our work provides a framework in the form of a waterfall model within which firewall systems and their components can be designed and evaluated.

We introduce a reference model that captures existing firewall technology and allows for an extension to networking technologies to which it was not applied previously. The essential components of the reference model are authentication, integrity assurance, access control, audit, and their enforcement. All components are governed by a centralized security policy, and they can be deployed in a distributed fashion to achieve scaling.

We introduce a formalism that is based on Hierarchical Colored Petri Nets (HCPN) to describe the functionality of mechanisms used by firewall technology. HCPNs provide us with a means of description, composition, simulation, and analysis of firewall systems.

The implementation of a firewall depends on its underlying network technologies. We describe the concept of authenticated signaling and report on the design, implementation, and exploration of its realization for asynchronous transfer mode (ATM) signaling, using above reference model. The resulting security mechanism can be used as a building block in the construction of firewall systems.

1. INTRODUCTION

Data communications networks have become an infrastructure resource for businesses, corporations, government agencies, and academic institutions. Computer networking, however, is not without risks as Howard ([How97]) illustrates in his analysis of over 4000 security incidents on the Internet between 1989 and 1995. Firewall technology is one mechanism to protect against network-based attack methods. A balanced approach to network protection draws from several other fields, such as physical security, personnel security, operations security, communication security, and social mechanisms ([ISV95, Part II]).

Classically, firewall technology has been applied to TCP/IP (*transmission control protocol, internet protocol*; [Pos81a, Pos81b]) internetworks (reviewed in chapter 3). Firewalls are used to guard and isolate connected segments of internetworks. “Inside” network domains are protected against “outside” untrusted networks, or parts of a network are protected against other parts. Various architectures for firewalls have been published and built, such as filtering routers, or application level proxy services (see section 3.1).

Landwehr suggests the application of formal models of security for secure system design (see [Lan81, §1]): by demonstrating that a design to which an implementation corresponds enforces a formal model of security, a convincing argument can be made that the system is secure. To date there is neither a reference model nor a theoretical background for firewall technology. There is also no definition of the term. A reference model of firewall technology is the first contribution of this dissertation.

A firewall system is implemented through a number of mechanisms that collectively achieve the desired functionality. This dissertation introduces a design tool approach based on *Hierarchical Colored Petri Nets* (HCPN, short CPN) to describe the functionality of such mechanisms. CPNs are a formalism that is suited for modeling a system in which synchronization, concurrency, composition, and activities on regulated flows of information are of concern ([Jen96a]). It can be used for the representation, combination, simulation, and analysis of firewall components and firewall systems. The introduction of this design approach is the second contribution of this dissertation.

The implementation of a firewall system depends on the underlying network technology because separate network technologies offer a variety of features and services. This dissertation investigates what aspects are common to a subset of thbib.c ese, namely connection-oriented networking technologies, that can be used to contribute to the provision of security services. We concentrate on architectural support rather than the high performance application of understood firewall techniques such as packet filtering. The third contribution of this dissertation is the report on the design of one such concept, named *authenticated signaling*[1], and the report on the design, implementation, and exploration of a prototype as proof of concept.

1.1 Thesis Statement

> It is possible to specify a reference model for firewall technology, given that firewall technology is viewed as a set of mechanisms that can enforce a network domain security policy on communication traffic entering or leaving a network policy domain. The reference model captures the state of the technology up to mid 1997 and allows for its extension to networking technologies to which it was not applied previously.
>
> A model based on Hierarchical Colored Petri Nets can be used to describe, compose, simulate, and analyze firewall components as well as firewall systems.

[1]This term contains the word *signaling* because connection management protocol messages are secured. It is called *authenticated* because connection authentication security services, i.e., authenticity *and* integrity of such signaling messages are provided.

> The signaling channel of connection management protocols can be used as a transport mechanism for security functions to secure the connection management itself and to provide a basic security building block to other components on the network.

1.2 Dissertation Outline

This dissertation is organized as follows. Chapter 1 gives an introduction and motivates the topic of this dissertation. Chapter 2 defines terminology used throughout this dissertation. Chapter 3 reviews previous work in network access control security mechanisms and firewall technology. Chapter 4 gives a framework for firewall technology in the form of a waterfall model of the firewall life cycle to place the contents of the remaining chapters into a context and to explain their relationship. Chapter 5 presents a reference model for firewall technology. Chapter 6 describes a design tool approach that is based on the formalism of Hierarchical Colored Petri Nets. Chapters 7 and 8 describe the concept of authenticated signaling and the design, implementation, and exploration of its realization for an ATM (*asynchronous transfer mode*) connection management protocol. We close with chapter 9, summarizing our experiences, presenting ideas for future directions, itemizing our contributions, and drawing some conclusions.

1.3 Chapter Summary

Firewall technology is a mechanism that protects computer networks as a resource as well as hosts connected to them. This dissertation contributes in three ways to the field of firewall technology: it provides a reference model, it introduces a design tool approach based on Hierarchical Colored Petri Nets, and it reports on the design of a network security mechanism that can be used for implementing firewall systems and the design, implementation, and exploration of a proof of concept realization thereof.

2. TERMINOLOGY

This chapter defines terminology used throughout the dissertation and gives our definition of the term *firewall technology* for the purpose of this dissertation. Technical terms not defined in this section are used according to their definitions in [BC94, Com96, LS87]. Definitions in this chapter are based in [BC94, Com96, LS87] but extended to fit our needs.

A *network* is a communication system that allows computers and other electronic devices attached to it to exchange data. A *router*, *gateway*, or *switch* is a device that attaches to two or more networks and forwards information from one network to another.

We define *communication traffic* to be the transmission of information over a network. We denote the set of all possible transmissions by $\mathbb{T}$. Any instance of communication traffic, called a *transmission unit*, is a tuple $(ctrl, data) = t \in \mathbb{T}$ consisting of control information ($ctrl$) and data ($data$) either of which may be empty, but not both. The interpretation of what amount of information comprises a transmission unit depends on the protocol layer of observation. For example, in an instance of network layer functionality (see *open systems interconnection* (OSI) model [DZ83]), the Internet Protocol ([Pos81a]), transmission units are called *datagrams*.

Attribute $t.ctrl$ can contain information, such as source ($t.ctrl.src$) and destination ($t.ctrl.dst$) addresses, reliability ($t.ctrl.reliab$) and flow control ($t.ctrl.flow$) information, access request information ($t.ctrl.acc$), and quality of service parameters ($t.ctrl.qos$). Attribute $t.data$ may contain application-specific payload or a payload that, at a higher layer of abstraction, can be interpreted as a transmission unit in itself. Transmission units do not need to contain all fields of t. For example, some fields may not be necessary at all, such as $t.data$ in control messages; others may be available through established state, such as $t.ctrl.qos$ in an existing connection.

A *security policy* is the definition of the security requirements for a given system. It can be defined as a set of standards, rules, or practices. We define a *network domain security policy* $\mathbb{P}$ as a subset of a security policy, addressing requirements for authenticity and integrity of communication traffic $t \in \mathbb{T}$, authorization requirements for access requests $req(t.ctrl.src, t.ctrl.dst, t.ctrl.acc)$ $\forall\ t \in \mathbb{T}$, and auditing requirements.

A *network policy domain* $\mathbb{D}$ is a set of interconnected networks, gateways, and hosts offering services that are governed by a network domain security policy $\mathbb{P}$.

Communication traffic is called *inside* (short, *in*) at a given time, if it is transiting or governed by systems or facilities within the network policy domain; otherwise it is called *outside* (short, *out*). Communication traffic is *entering*, if it is moving from the outside to the inside of a network policy domain. It is *leaving*, if it is moving from the inside to the outside.

Using the above defined terminology and a study of firewall systems as described in chapter 3 we arrive at the following characterization of the term *firewall technology*:

> *Firewall technology* is a set of mechanisms that can enforce a network domain security policy $\mathbb{P}$ on communication traffic $\mathbb{T}$ entering or leaving a network policy domain $\mathbb{D}$.
>
> A *firewall system*, or *firewall*, is an instantiation of firewall technology.

2.1 Remarks

Our definition covers the state of firewall technology up to mid 1997. Furthermore, it includes the view of firewall technology as a distributed security architecture placed on the data transmission path between communication endpoints.

Our definition of firewall technology states that communication traffic needs to *enter or leave* a network security domain to be of interest to firewall technology.

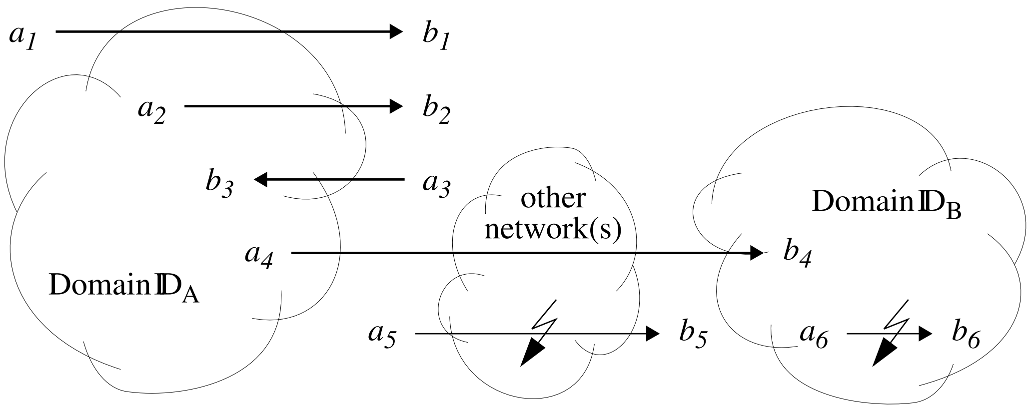

Figure 2.1 Communication traffic governed by firewall technology between sender a_i and receiver b_i. Firewall technology can be applied to communication traffic for $i \in \{1, 2, 3, 4\}$, but not for $i \in \{5, 6\}$.

Figure 2.1 illustrates the possible combinations for point-to-point communication. For any traffic between sender a_i and receiver b_i the definition includes traffic that traverses the protected domain $\mathbb{D}_A$ (i.e., $\{a_i, b_i\} \notin \mathbb{D}_A$; $i = 1$) and traffic that traverses networks that are not part of $\mathbb{D}_A$ with $a_i \in \mathbb{D}_A$ and $b_i \notin \mathbb{D}_A$ (outbound traffic; $i = 2$), $a_i \notin \mathbb{D}_A$ and $b_i \in \mathbb{D}_A$ (inbound traffic; $i = 3$), or both $a_i \in \mathbb{D}_A$ and $b_i \in \mathbb{D}_B$ (virtual private networking between $\mathbb{D}_A$ and $\mathbb{D}_B$; $i = 4$). Communication traffic between a_i and b_i that neither enters nor leaves a network policy domain is not subject to firewall technology ($i \in \{5, 6\}$).

Chapter 3 reviews examples of existing firewall components, such as packet filtering, network address translation, generic circuit level forwarding, and application-specific proxy forwarding. It is a practical challenge to the designer and maintainer of a firewall system to ensure that their functionality implements the security policy.

Mechanisms, such as packet filtering or network address translation, can be examined by various criteria to determine their possible contribution to firewall technology, such as trust assumptions, at which layer of abstraction they operate, what performance characteristics they have, and if they appear transparent to users. The final appearance of each individual firewall is determined by its designer's choices in this space of criteria.

A first example of distinguishing criteria is the difference in how much trust a mechanism assumes. For example, mechanisms that operate on received data without verification of its authenticity or integrity make stronger assumptions of trust than mechanisms that use strong cryptography[1] to verify the authenticity of control packets. Assumptions of trust that are present, but that are not warranted, can result in less secure firewall systems.

Secondly, each mechanism operates at a single layer or a range of layers of abstraction (*operating range*) in the layered model of networking ([DZ83]). For example, the closer the operating range to the application layer the more application-specific detailed information is interpretable by the mechanism, which allows for the implementation of fine-grained access control decisions and authentication of high-level identities.

Firewall designers must understand the assumptions, functionality, and shortcomings of used mechanisms, as well as issues related to their interaction, such as name resolution, communication, and caching. The set of mechanisms can then be described by their functionality, their interaction with other mechanisms, and their interaction with systems outside the firewall itself.

2.2 Chapter Summary

This chapter defines terminology used throughout the dissertation. It provides our definition for firewall technology to be a set of mechanisms that can enforce a network domain security policy on communication traffic entering or leaving a network policy domain.

[1] The term *strong cryptography* (e.g., strong authentication) is used to indicate that algorithms, key sizes, and parameters are used that make it computationally infeasible by the technology available at a given time to break the cryptographic protection mechanisms by brute force attempts.

3. RELATED WORK

This chapter presents an overview of mechanisms that are used for building firewalls. A study and comparison of these mechanisms resulted in our definition of the term "firewall technology" in chapter 2.

Furthermore, this chapter describes characteristics of a particular high performance networking technology, the *asynchronous transfer mode* (ATM), that need to be addressed for the provision of firewall security services. Related research by others is presented.

3.1 Security Mechanisms in Traditional Firewall Technology

Firewalls are implemented using a variety of security mechanisms, such as packet filtering, packet labeling, network address translation, and proxy forwarding. Several research papers and some textbooks describe various, differing approaches (see for example, [GS96, §21], [BC94], [CZ95], [SH95], [Ran92], [Ran93c], [Ran93b], [RLCB94], [AR94b], [AR94a], and [Atk95a]). A subset of these mechanisms may interact to make up a comprehensive firewall system.

3.1.1 Packet Filtering

In packet filtering routers security policies are translated into lists of rules (see [BGP+94], [Cha92], [Dig92]). Each rule allows or denies packets through the firewall based on some semantic interpretation of the packets' contents. Rules may interact with each other, e.g., through their order. If no rule is applicable a *default deny stance* may be taken (e.g., the action "discard packet" can be performed). This approach is an example of the implementation of a concept in computer security that, to the extent possible, systems should be failsafe ([CZ95]).

In a TCP/IP (*transmission control protocol, internet protocol*) packet filtering firewall, datagrams that arrive at the router are passed to a packet filtering mechanism. The filter calculates a decision to discard or forward each packet according to specified rules based on the packet header and content, for example, source and destination addresses and port numbers, and possibly a saved state indicator, for example, flow identifiers. Subsequently, the filter then enforces each decision. The rules operate exclusively on the contents of the datagram because no context is maintained across datagrams that belong to the same connection.

Stateful packet filters do save state for packets that belong to the same session (e.g., an ftp session). This approach allows packet filters to make decisions within the context of a particular connection. The ftp protocol can serve as an example: inbound ftp data connections can now selectively be allowed through packet filters only when necessary instead of allowing all incoming ftp data connections through the packet filter as was done previously.

Although a packet filter offers the opportunity to handle and verify all data passing through it, the lack of end-to-end context prevents a security association from being established. Packet filtering does not provide integrity and authenticity control of the examined packets. The application of filtering rules to each datagram introduces some delay because their processing takes time. It may introduce jitter because the calculation of filtering results can introduce differing amounts of delay for separate packets.

3.1.2 Network Address Translation

Network address translation (NAT) is a mechanism that was originally proposed as a short-term solution for IP address depletion ([EF94]). Network address translators are placed at the borders of "stub" network domains. Each NAT device has a table consisting of pairs of local IP addresses and globally unique addresses. For all routed datagrams it translates local addresses into their associated globally unique addresses and vice versa. The association can be static or dynamic. NAT discards end-to-end significance of addresses, making up for it with increased state in the network.

NAT has the feature that the internal address space and network topology are hidden from the outside. Furthermore, except for address translation, NAT devices have few other network-based services enabled and are therefore less susceptible to attack than classical routers.

If an organization has several computer networks at physically separate locations and a firewall for each location, it can use these firewalls to automatically encrypt all traffic that is exchanged between them. This approach is called *virtual private networking* (VPN) and is difficult to achieve with NAT technology unless tunneling between participating NAT devices is provided. NAT does not work if higher layer protocols or applications use and expose the hidden local addresses (for example, DNS; [Moc87a, Moc87b]). NAT devices can prevent this problem from occurring in some cases by rewriting higher layer protocol messages with appropriately mapped addresses (for example electronic mail; [Cro82]). Applications that carry and use local addresses across a NAT boundary will not work unless the NAT device is capable of detecting and processing such instances ([EF94]).

3.1.3 Generic Circuit Level Forwarding

Circuit level firewalls group packets into connections, for example TCP connections, by maintaining state across packets ([KK94], [AMP96], and [LGL+96]). One way to achieve this association is by inserting a generic transport layer proxy process in the connection. Inbound as well as outbound connections must connect to the proxy process first before any data can be relayed. The proxy uses access rules to determine if the connection should be established or blocked. Circuit level gateways can provide elaborate access control mechanisms including authentication and additional client/proxy protocol message exchanges ([AMP96]).

An implementation of generic circuit level forwarding can be hidden in low-level libraries without the necessity of modification of client/server source code if user interaction is not required ([KK94]). Programs initiating connections may need to be modified for the provision of authentication information. Only few changes are necessary, but there are challenges, such as the availability of source code, the heterogeneity of system platforms, the distribution of programs, and the education of the user population. The ISI tunnel is an example of generic circuit level forwarding ([DC93]).

3.1.4 Application-Specific Proxy Forwarding

Application level firewalls (e.g., [Che92]) can interpret data in packets according to particular application protocols and add security services. They are circuit level proxies as described in section 3.1.3 with the capability of interacting with the communication endpoints through the application's high-level protocol. These proxies are application-specific: for each new application a separate forwarding service must be provided. These forwarding services can add fine-grained authentication and access control services to applications because of their capability to interpret the application protocol in use ([AR94a]).

Application-specific proxies add an additional single point of failure to services. Proxy services are usually small, service-dependent programs that can be scrutinized more easily for vulnerabilities before deployment than can server programs. An example of a set of proxy servers built according to this philosophy is the TIS firewall toolkit ([AR94b]). The drawbacks regarding the requirement of client software modification for generic proxies (section 3.1.3) apply to application-specific proxies. Each application-specific proxy adds protocol processing overhead of two additional round trips through the protocol stack.

3.1.5 Cryptographic Security Mechanisms

The following two cryptographic security mechanisms are standardized by the *Internet Engineering Task Force* (IETF) for the IP layer. We are using them here as examples of a class of cryptographic mechanisms used in firewall technology: the *authentication header* (AH), which provides integrity and authentication without confidentiality ([Atk95b]), and the *encapsulating security payload* (ESP), which provides confidentiality and optionally integrity and authentication ([Atk95c]). Both mechanisms can operate between a set of hosts and/or gateways, i.e., end-to-end, end-to-intermediate, or intermediate-to-intermediate, in unicast and multicast mode.

These mechanisms are designed to protect communication traffic against eavesdropping, unnoticed modification, insertion, or deletion. They can be used to build virtual private networks across untrusted networks, such as the Internet. No protection against traffic analysis attacks is provided ([Den82, §3.5.1]). They are usually algorithm independent by way of algorithm identifiers included as part of the security protocol. Standard default algorithms are specified to ensure interoperability of all implementations at a common denominator.

The mechanisms require the establishment of a *security association* (SA) among the set of hosts and/or gateways that are party to the protected communications. A security association is "the set of security information relating to a given network connection or a set of connections" ([Atk95a, §1.1]). For the application of these mechanisms there are practical concerns, such as key management, and efficiency considerations, such as the introduction of protocol processing overhead and the increase in communication latency.

The encryption of communication traffic in the presence of packet filtering is an example where two security mechanisms can work against each other. If the packet filter is not party to the SA used by the encryption mechanisms it cannot perform its function because it cannot decrypt the encrypted higher layer protocol headers needed to process its filtering rules.

3.2 Advantages of Firewall Technology

As the previous sections describe, firewalls can protect deployed computing systems and networked applications. Proponents argue that firewall technology is more than a retrofit patch for shortcomings in systems and protocol design (a survey conducted by the *National Computer Security Association* (NCSA) documents the positive experiences and perception of a small set of American businesses ([NCS97a]).) Because of their placement at the network perimeter, firewalls can serve as a centralized focus of security policy and as a place to collect comprehensive security audits, even in the presence of secure hosts. Firewalls address some problems of network security that cannot be addressed by host security mechanisms: they protect the network as a resource as well as the hosts connected to it and provide protection against some denial of service attacks ([SKK$^+$97, §4.4]).

The aggregation of security functions in firewalls allows for a simplification of management, installation, and configuration of security functions ([BCCH94]). They improve administrative control and network management via controlled exposure of internal network structure, topological flexibility, and transparency to the user ([BCCH94]). Security firewalls represent a technology that is widely accepted, available, cost effective, and economically justifiable to management personnel in charge of purchasing decisions ([NCS97a]).

3.3 Disadvantages of Firewall Technology

Conversely, firewall technology can provide a false sense of security: it may lead to lax security within the firewall perimeter (see [BCCH94, §3]), similar to the way the supposedly impregnable Maginot Line[1] led French army leaders to ignore the need for provision of additional defense mechanisms further inside their country ([Che97]). In [BCCH94, §3.1.1] this concern is expressed through another analogy: firewalls provide "a hard, crunchy outside with a soft chewy center."

Security firewalls neither provide "perfect security" nor are free of operational difficulties. They do not protect against malicious insiders. There is no protection

[1] After André Maginot (1877-1932), French minister of war. The Maginot Line was a 150 mile long system of heavy fortifications at the eastern frontier of France built before World War II to protect French territory from Germany. Germany did invade France again, but it went around the Maginot Line to do so. The Line itself was never taken by force.

against connections that circumvent the firewall, such as unauthorized modems attached to computers inside the firewall because the enforcing mechanism is bypassed. There is limited protection against *illicit rendez-vous* (unauthorized tunneled connections) and data-driven attacks, such as malicious executable code in downloaded Java applets ([BC94]). Because typical practice does not provide a check of firewall system configuration against the security policy, changes in system configurations may produce security holes ([NCS97a]).

Firewall technology has been developed for and applied to TCP/IP networks almost exclusively ([BCCH94]). It was never developed according to a reference model and only addressed acute problems at hand. Because of the reactive character of firewall design, there is little reason to expect that effective protection against new attacks is guaranteed. An incentive for advances in the state of the art of firewall technology has been the need to develop defenses against attack scenarios that have initially succeeded through or against firewalls.

3.4 Security Mechanisms for High Performance Network Technologies

The implementation of firewalls depends on its underlying networking technologies because they offer a variety of features and services. This dissertation addresses the search for mechanisms that can provide firewall security services for high performance networking technologies.

This section describes why security services need to be provided in ATM, before it lists a set of related research projects and product development efforts in the field. The security mechanisms contributed by these projects can be used to construct firewall systems connected to high performance networking technologies.

3.4.1 Problems for Security Services in High Performance Network Technologies

There are a number of problems related to the high speed cell relay nature of the asynchronous transfer mode and its small cell size. These problems are the reasons why existing firewall mechanisms are insufficient to provide firewall security services for ATM networks.

Security services on ATM data traffic (in contrast to ATM control traffic) must be capable of keeping up with the gigabit speeds at which cells are transmitted. If cryptographic mechanisms are used, they need to operate at the hundreds of megabits per second or even gigabit per second speeds. This requirement includes not only encryption and decryption but also crypto synchronization and key update/exchange. Such mechanisms need to introduce as little delay as possible to minimize their impact on the provisioning of quality of service requirements by the network.

Cryptographic mechanisms need to interoperate at various different rates if, for example, a client connects via a *synchronous optical network* (SONET) interface at OC-3c (*optical carrier*; 155 Mb/s) to a network to which a server may be connected via an OC-48c (2.488 Gb/s) interface (see [Pie96]). Furthermore, they require security messages to be exchanged that are of considerable size (on the order of tens to thousands of bytes). ATM cells of 48 byte payload each cannot carry those messages without going through additional encapsulation and segmentation/reassembly steps.

Most practical ciphers have the property that the same cryptographic key should be used only for a limited time because the more ciphertext is observed by a passive wiretapper the higher the likelihood that the key can be broken. There are results by Claude Shannon and Martin Hellman that determine how much ciphertext needs to be observed before a cipher is theoretically breakable (see e.g., [Den82, §1.4.3] for a discussion). The practical importance of these theoretical results is limited by the observation that the interception of enough ciphertext still does not mean it is a computationally tractable problem to break the cipher. Nevertheless, high speed communications cause more ciphertext to be observable in a given amount of time than at slower speeds. It is therefore necessary to change keys more often to limit the amount of ciphertext a cryptanalyst can collect under one key.

It is desirable to provide fine-grained security services to clients. An example is the encryption of different virtual circuits by different keys to grant all users confidentiality. Such fine-grained multiplexing requires key-agility (see also section 3.4.3), the application of keys that belong to the different circuits.

3.4.2 High Performance Packet Filtering

Hughes et al. at Network Systems Corporation have designed and built a high performance packet filter to satisfy performance requirements of ATM's high speed transport ([Hug96]). It implements an extension of firewall packet filtering practice at higher speeds through hardware assistance. It operates by placing a representation of a given policy close to the data path in switches. For each cell (and if necessary for the reassembled higher layer frame) it is determined if the policy allows forwarding or requires discarding of the cell. *Content addressable memories* (CAMs) are used to hold the information that describes cached policies. The cache is divided into three parts: to ensure only authorized virtual circuits are used, to ensure reassembled IP datagrams meet their classical packet filtering policy, and to ensure socket policies are valid.

The mechanism does not introduce a new or improved concept for security enforcement. It is merely an application of packet filtering (section 3.1.1) at higher speeds through hardware assistance. With the technology available in mid 1997 the mechanism cannot be used at data rates higher than 155 Mb/s ([Hug96]) while network technologies of higher data rates are already available. We expect that the performance development of packet filters will continue to lag behind that of high performance networking technologies for some time to come.

3.4.3 High Performance Key-Agile Encryption and Cryptographic Synchronization

Research efforts at Sandia National Laboratories, MCNC, Portland State University, and the University of Cambridge, UK have focused on the design, implementation, and evaluation of experimental key-agile cryptographic systems for ATM networks. [TPB+96], [SHB95], [SBHW95], and [Chu96] report on high speed encryption methods, high performance key-agility techniques, and methods for cryptographic system synchronization. The focus in [TPB+96] is on the exploration of the feasibility of a hardware encryptor/decryptor prototype that can perform at OC-3 speeds. A key-agile device was built that uses a number of *linear feedback shift registers* (LFSR) in parallel to produce linear recurring sequences with long periods as a key stream for encryption and decryption ([TPB+96]).

The research and development efforts by these groups demonstrate that privacy and integrity services for data (in contrast to control information) can be provided in high performance network technologies at the mentioned speeds. The same critique as in section 3.4.2 applies: the performance of these mechanisms lags behind the performance development of high performance networking technologies.

3.4.4 Signaling Support for Authentication

In a standards contribution, Lyles ([Lyl94]) motivates the development of authenticated signaling with two examples: fraud in a highly deregulated and competitive telecommunications environment, and the introduction of unacceptable performance bottlenecks for network access control if typical Internet security mechanisms are applied. The idea is to address the problem with modifications to connection management protocols to provide a basic building block for security services. The primary drawback of this approach is that it requires changes to existing and deployed technologies. However, once developed it can be applied to other connection management protocols.

Smith and Stidd ([SS94]) proposes concrete solutions to the problems of user authentication and billing for services and products provided by end systems in *broadband integrated services digital networks* (B-ISDN). Their proposal recommends the introduction of an information element in point-to-point and point-to-multipoint connection establishment messages. The proposal does not contain details regarding the types of information that need to be present in the information elements. Furthermore, it does not mention the need for the authentication of further control information, such as connection release messages to protect against the threat of a denial of service ([GS96, §25]).

3.4.5 Signaling Support for Other Security Services

Tarman et al. at Sandia National Laboratories ([TPB+96] and [Pie96]) have investigated techniques for integrating security enhancements within standard ATM protocols. Their work has focused on hardware and software encryption in high performance networks (see section 3.4.3) as well as signaling support for encryption, authentication, and key exchange. Their work in authenticated signaling goes further than [SS94]. They describe requirements for, and an example of, an authentication information element ([TPB+96, §8]). Tarman et al. state the need for an "identity validation" message that may occur at any time during a connection's lifetime. They do not address the threat of denial of service through unauthenticated connection teardown. [TPB+96] does not concentrate on the issue of network layer access control — a central focus of our research.

3.5 Chapter Summary

Firewall technology is based on a combination of mechanisms, such as packet filtering, network address translation, circuit level forwarding, and application layer proxy relaying. As many other technologies, firewall technology has advantages, as well as disadvantages.

The existence of high performance networking technologies has led to the development of security mechanisms that can be used in the design and implementation of firewalls, such as high performance packet filtering, key-agile encryption and synchronization, and signaling support for security services. They address problems specific to their technologies and provide generic security services for firewall technology.

4. THE FIREWALL LIFE CYCLE

Firewall systems undergo a gradual development and evolution, called a life cycle: figure 4.1 is our derivation of the waterfall software life cycle model that goes back to the late 1950s (see [GJM91] for a description), applied to firewall technology. The original waterfall model was a result of the experiences gained in the development of a large software system, named SAGE.

Figure 4.1 illustrates the firewall life cycle. The *phases* (on the left in boxes; section 4.2) use *methods* (on the right in italic font; section 4.3) to produce the *deliverables*, which are the output of phases (in the middle in bold face font). The life cycle does not terminate, but continues through a periodic cycle to review possible modifications to the system or changes in the environment of the system ([GJM91]). The arrows pointing upwards in the figure illustrate that any earlier phase can be revisited occasionally, for example if a new, improved firewall component becomes available that is to be incorporated into an existing firewall system, or if a phase uncovers defects in a previous phase. A phase does not necessarily have to be completed before the next one begins, but successive phases may overlap.

We use this waterfall model as a framework for firewall system design. It serves to place the contents of the following chapters of this dissertation into the context of the firewall design life cycle.

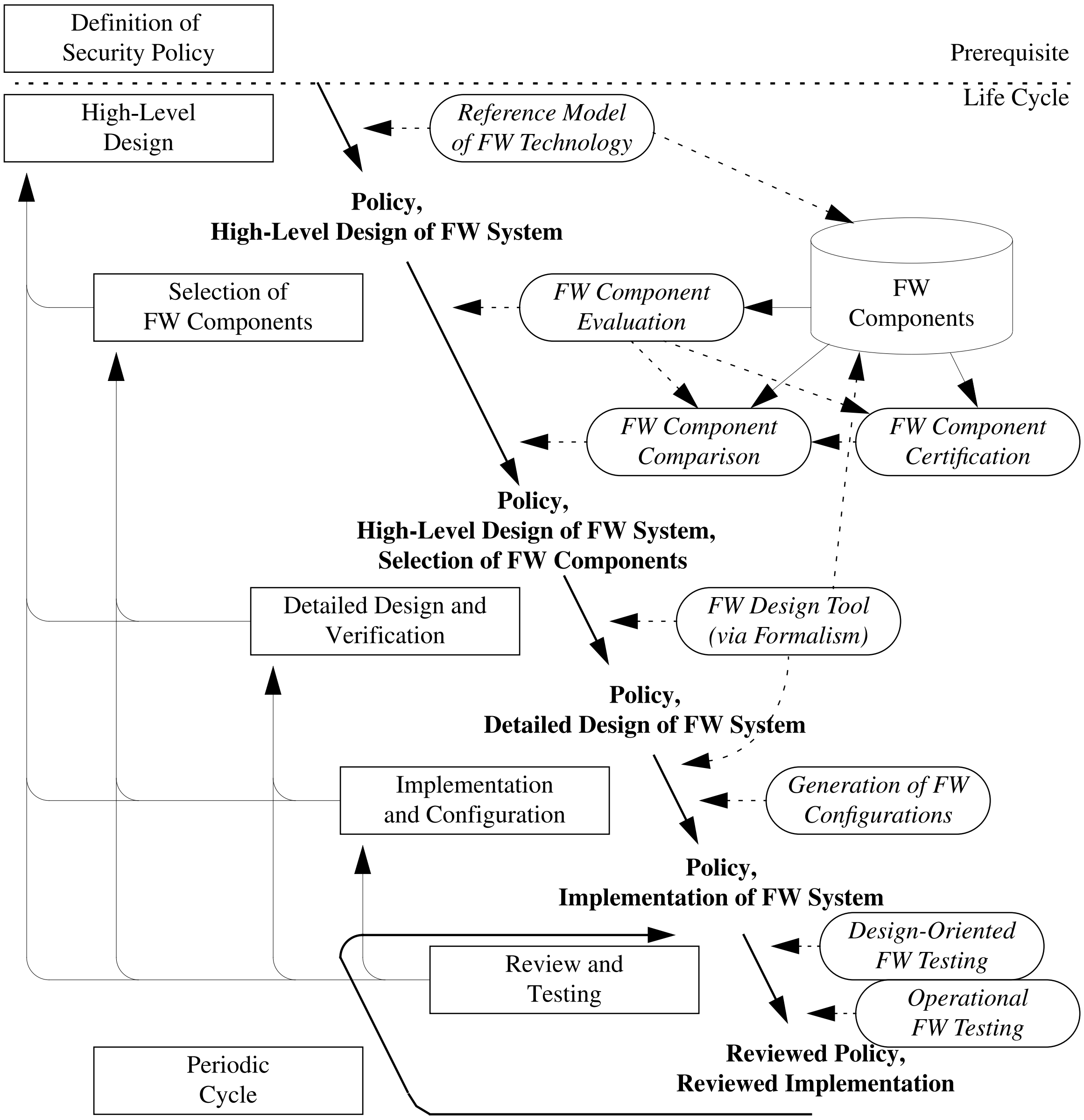

Figure 4.1 The firewall life cycle. This waterfall model consists of *phases*, *deliverables*, and *methods*. The phases (on the left in boxes) use methods (on the right in italic font) to produce the deliverables between the phases (in the middle in bold face font). The life cycle does not terminate, but continues through a periodic cycle to review possible modifications to the system or changes in the environment of the system. The arrows pointing upwards indicate feedback by phases to their predecessors. The database on the right side illustrates the set of all available firewall components.

4.1 Example

This section presents an example of how an organization that designs a firewall system from the beginning would follow the firewall life cycle model. The following sections describe the various phases that are present (section 4.2) and the various methods that are available to firewall system designers during this process (section 4.3). The description of the example refers to figure 4.1.

An organization that designs a firewall first needs to define its network domain security policy. The designer then decides on a high-level design for its firewall system following the reference model for firewall technology as defined in chapter 5.

The selection of firewall components for the detailed design later can benefit from several methods. The designer can browse a comprehensive list of firewall products (firewall component comparison; cf. section 4.3.4) to find components that are likely to be of value, possibly giving preference to certified products in that list (firewall component certification; cf. section 4.3.3). Before the final selection is made, the designer can evaluate some of the firewall components himself (firewall component evaluation; cf. section 4.3.2).

The designer would then use those components and specify a low-level design of the firewall system. To date there has been little help available in this phase of the life cycle. The design tool and formalism for firewall mechanisms and systems as described and analyzed in chapter 6 can be used as the basis for firewall design tools. It offers support for the graphical design of firewall systems using descriptions of firewall components from a library, simulations of the behavior of the designed system, and the formal verification of certain properties of interest. Furthermore, the design tool may be used to generate configurations or software components for the firewall in its implementation and configuration phase.

Next, the organization would consult an expert to review the design through design-oriented testing. After deployment, firewall testing would be used periodically to revalidate confidence in the correctness of the firewall installation, taking corrective actions when they become necessary.

4.2 Phases

The following phases are present in the firewall life cycle, as illustrated in figure 4.1. More detailed explanations of these generic phases as applied to software development can be found in software engineering books (for example, [GJM91]).

4.2.1 Definition of Security Policy

The prerequisite phase of the model is the definition of a network domain security policy. A security policy includes answers to questions, such as what the network perimeter being protected is, which network-based computer services should be available to outside entities, who those entities are, which outside services should be available to users located inside the protected network domain, what the necessary controls are, what the security impact of services is, and what the assumptions on service and system behavior are.

Such a policy is a prerequisite for the successive stages of the model. The development, representation, analysis, or management of security policies is addressed by the work of others (for example, [Dij96], [Smi93, §6], and [Woo94]). It is outside the scope of this dissertation. The deliverable of this phase is the policy.

4.2.2 High-Level Design

The purpose of the high-level (or architectural) design is to specify the overall module structure and organization rather than details. The deliverable of this phase is a high-level design of the firewall system.

4.2.3 Selection of Firewall Components

Once the high-level design is documented, designers can choose among various firewall components or modules to construct the firewall. This phase allows designers to make a selection of the components that may be part of the firewall system. The deliverable of this phase is a set of firewall components that may be used in the next phase.

4.2.4 Detailed Design and Verification

During detailed design the selected components are used to architect a concrete firewall system in accordance with its high-level design. The components alone are not sufficient, and modules need to be designed, operational procedures need to be specified, and network reconfigurations need to be defined. The deliverable of this phase is a detailed design.

4.2.5 Implementation and Configuration

During this phase the firewall system is built and configured. The deliverable of this phase is an operating firewall system.

4.2.6 Review and Testing

Once the firewall system has been built and deployed, it needs to be tested to validate that it is capable of enforcing the desired network domain security policy. This phase reviews not only the firewall system, but also the policy that is being enforced. The deliverables of this phase are a reviewed and improved network domain security policy and a reviewed and improved implementation of the firewall system.

4.2.7 Periodic Cycle

A firewall system is subject to frequent changes (e.g., its internal configuration, its network configuration, or the network domain security policy it is built to enforce).

The purpose of this phase is to revisit a firewall system periodically to assure that the firewall system still operates as it is supposed to, similar to the concept of *preventive maintenance* (PM) used in engineering disciplines to establish a periodic, proactive review of the system. The deliverables of this phase are a reviewed and improved network domain security policy and a reviewed and improved implementation of the firewall system.

4.3 Methods

Figure 4.1 and the example in section 4.1 illustrate when the various methods are applied in the firewall life cycle. This section motivates and explains these methods at a high level. We point out which of these methods are examined in forthcoming chapters of this dissertation. This section serves as an overview of the benefits of those methods and to place them in the overall framework set by the waterfall model of the firewall life cycle.

4.3.1 Application of Reference Model for Firewall Technology

A *security model* is a description used in analyzing or explaining the desired behavior of the security-relevant portions of a system ([Lan81]). Chapter 5 presents such a security model: the reference model for firewall technology.

The goal of this functional model is not to produce a standard so that conformant implementations can interoperate. Rather, the goal is a functional description arrived at from a study of the application domain (described in chapter 3) to facilitate understanding of the concepts of network access control and how they interact. The model gives a view of this functionality to the extent that it is enforceable in the part of the network which is under local control, the network policy domain.

The purpose of the reference model is to give an understanding of

- which functions may need to be present in a firewall system,
- how they need to be enforced,

- how they interact on a conceptual level,
- how their distribution can yield scaling benefits,
- how they can be applied at various protocol layers, and
- how they can be composed into an overall firewall security architecture.

Chapters 7 and 8 illustrate how a reference model can guide the design and implementation of security mechanisms through the example of authenticated signaling.

4.3.2 Firewall Component Evaluation

This method generates evaluations of firewall components. The analysis of network security products takes expertise and time. Its goal is to facilitate component certification (section 4.3.3) and component comparison (section 4.3.4). A typical component evaluation as described in [CSI97] or [SSS$^+$] uses the following categories for its analysis:

- Product Identification
- Documentation and User Education
- Functionality
- Operations and Maintenance
- Cost

Each category consists of a set of questions, criteria, or experiments, any one of which may or may not be applicable to the firewall product under investigation. By systematically examining a firewall product with these criteria, one creates a comprehensive understanding of the quality of the product. Some questions can be answered by consulting marketing material of products; others may require detailed research and specialized expert knowledge in a variety of fields.

4.3.3 Firewall Component Certification

The idea behind firewall component certification is to evaluate firewall products and systems using an established list of tests and procedures. Firewalls receive certification if they pass the tests. The goal of certification is to reduce real and perceived risks for computer systems protected by firewalls through the assurance that the components operate at or above the certified level of quality. One example is the laboratory for a certification program at the *National Computer Security Association* (NCSA) ([NCS97b]).

The idea of certification has the drawback that products are tested in a generic, artificial laboratory environment, which is different from most real operational environments. For example, it is impractical to assume that all problems introduced by end-user customization can be anticipated by such tests.

4.3.4 Firewall Component Comparison

This method is a comparative study of firewall components. A popular representation of its results is a matrix in which firewall components are listed on the one axis (rows, for example), and criteria are listed on the other axis (columns, for example). The squares in the matrix contain the evaluation result of the criterion listed in its column applied to the firewall component listed in its row. The purpose of such a matrix is to provide a representation for overview and comparison of firewall components and their criteria.

There is at least one such comparative study available that is periodically updated and published: the firewall product matrix by the *Computer Security Institute* (CSI) ([CSI97]).

4.3.5 Application of Firewall Design Tools

Chapter 6 explains a new approach to the design and analysis of firewall mechanisms and firewall systems. It is based on a formalism called Hierarchical Colored

Petri Nets. Chapter 6 explains how the formalism can be used as the theoretical foundation for a firewall design tool.

The approach offers the exploration of the functionality of firewall mechanisms. It offers the online design of complex firewall systems through the combination of firewall components that are expressed in the same formalism. The behavior and properties of designed systems can be examined through on-line simulation. Furthermore, results in Applied Petri Net Theory can be used for the exploration of theoretical properties of the modeled systems.

4.3.6 Generation of Firewall Implementations and Configurations

Design tools as mentioned in section 4.3.5 can support the automatic generation of the implementation of components. Using this same mechanism, Pinci and Shapiro ([PS91]) reports the development of a software application for electronic fund transfer. Calabrese ([Cal96]) describes a tool for building firewall-router configurations, thus demonstrating the possibility of providing portability of high-level policies across platforms. Furthermore, Calabrese's approach is a solution to overcome a difficulty described by Chapman in [Cha92] where he concludes that the usage of contemporary firewall configuration languages is a process prone to error.

4.3.7 Design-Oriented Firewall Testing

Ranum proposes an approach called design-oriented testing of firewalls ([Ran96, Ran97]). It is a combination of manual and automated testing of configurations and release levels of the deployed firewall components and accessible network-based services. Design-oriented testing examines firewalls first at a high-level and then top down at increasingly detailed lower levels as far as it is sensible to go.

It proceeds in the following steps:

1. Comparison of the implementation to the owner's plan of what is supposed to be implemented.

2. Manual investigation of the configuration through management interfaces.

3. Operational testing of the firewall. The analyst would use tools to probe the firewall and network behind the firewall for exposed services and would test if the filtering rules perform the actions they are supposed to perform. Section 4.3.8 goes into more detail of this step.

4. Examination of the allowed services. The analyst needs to assure that all available patches to known vulnerabilities are applied.

5. Assignment of a periodic review cycle of the firewall by the same process.

The approach results in:

- A *policy review.*

- An *implementation review*, which consists of an approved firewall configuration and component release levels.

- An *assessment of services*, which consists of an approved list of services, their configurations, and their release levels.

- A *review cycle.*

This approach is applicable at the point where a firewall implementation already exists. It does not give guidance for how to design, choose, build, or deploy a firewall. A further discussion of this issue is outside the scope of our work.

4.3.8 Operational Testing of the Firewall:

Operational testing is known under various names, such as "penetration testing," "firewall testing," or even the "tiger team approach," and promoted by a number of practitioners (see for example [MS96, Sch97]). The idea behind this approach is to probe deployed systems for vulnerabilities: a black-box evaluation of the firewall after its installation is performed. Firewall testing methods have advantages and drawbacks analogous to those present in software testing: one can ensure proper behavior of the system in certain common scenarios and situations, possibly discover vulnerabilities, but one can never be sure to have tested enough and not missed a major flaw ([GJM91]).

The major advantage of firewall testing is that it can be applied periodically as an effective means for revalidating confidence in one's firewall installation ([MS96]). This approach addresses the concerns of system administrators to at least periodically assure the correct behavior of firewall systems ([IV97]). Ranum mentions that services are part of the firewall system to the extent to which they serve security functionality. They are subject to review as much as the other parts of the system.

4.4 Chapter Summary

A variety of methods is used during the phases of the life cycle of a firewall system for its design, implementation, and maintenance. This chapter motivates the contents of the remaining chapters at a high level and places them in a waterfall model of the firewall life cycle.

5. REFERENCE MODEL FOR FIREWALL TECHNOLOGY

Chapter 4 presented a waterfall model of the firewall life cycle. It motivated the existence of a reference model for firewall technology and placed it into an overall context.

This chapter presents such a reference model for firewall technology. An earlier version of the model was presented in [LS96a] (see also [LS96b]). Computer networking is based on a layered model of communication. Communication protocols are distributed algorithms that execute between peer instances of the same layer or a range of layers. Similarly, the reference model for firewall security services as described in section 5.1 applies to a single layer or a range of layers. The reference model can be applied repeatedly at several layers within a network system as described in section 5.6.

The model is descriptive in that it captures the functionality of all firewall components and systems we have examined during the development of the model (see section 3.1). It is prescriptive because its components are sufficient to provide desired network access control security services (see section 5.2 for arguments why the various components are sufficient). Its prescriptive character can be used as a guideline for system design.

In general, the analysis, manipulation, and simulation of a modeled system can lead to new knowledge and insight without the risk, cost, or inconvenience associated with its direct manipulation ([Jen96a]). The process of modeling a system gives the modeler an improved understanding of the modeled system: Jensen states that modeling as an educational tool is often its primary benefit ([Jen96a, §1.7]). One of the limitations system developers face is their own inability to cope with too many details at the same time: models help overcome this limitation ([Jen96a]) and can prove beneficial during a system's implementation (as experienced in [SSK98, §4.1]).

The main benefits of the reference model are the provision of an understanding of

- which functions may need to be present in a firewall system,
- how they need to be enforced,
- how they interact on a conceptual level,
- how their distribution can yield scaling benefits,
- how they can be applied at various protocol layers, and
- how they can be composed into an overall firewall security architecture.

5.1 Reference Model

The reference model focuses on functionality required by firewall systems to enforce network domain security policies. For that reason we chose a functional model over other types of models, such as data processing, classification, stimulus-response, or process models ([GJM91]). The idea is that systems are, at a conceptual layer, composed of separate, interacting functional components.

Our reference model can be interpreted as a system composed of several types of security components. The components are combined under certain constraints to make up a firewall system. The components interact with each other and the rest of the system, for example through functional interfaces or the sharing of state information. The interaction with the rest of the system is not depicted in the model. Section 5.2 describes the components in detail.

Figure 5.1 displays a high-level view of the reference model of firewall technology (a more detailed representation is presented subsequently in figure 5.2). The representation in figure 5.1 is used in this dissertation to explain the reference model in a stepwise refined manner and as a simplified representation of the model for presentation purposes.

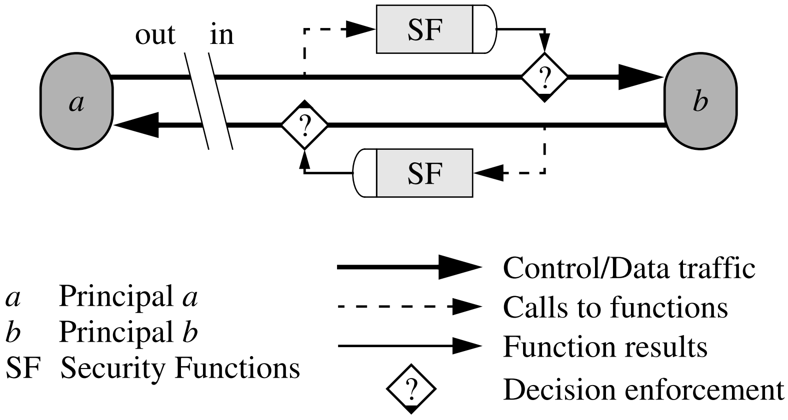

Figure 5.1 Abbreviated version of the reference model displayed in figure 5.2. *Security functions* (SF) are enforced on inbound and outbound communication traffic between principals a and b.

Consider the case where a principal a outside of a protected network policy domain attempts to communicate with a principal b inside that domain. The gap between “out” and “in” can be filled with intermediate networks of any technology and topology so long as data can be transmitted between the sender’s and the receiver’s networks. Everything between the gap and the representation of principal b is considered part of the protected network policy domain.

All communications are divided into transmission units that are transmitted by the network. The reference model operates on one protocol layer or a range of protocol layers on transmission units at that layer (range, respectively). It operates on inbound as well as outbound communication traffic. Transmission units are handled separately by the firewall, though state may be retained. The heavy, solid line represents the conceptual path that transmission units travel.

Shaded boxes represent functions. In figure 5.1 the boxes labeled SF represent a collection of *security functions* that are applied to transmission units exchanged between principals a and b. The dashed arrows represent the invocation of this collective function SF. Each SF receives portions or possibly even (a copy of) the entire transmission unit as input arguments. SFs calculate a result PASS or FAIL for each transmission unit. The diamond with the question mark (◇?) represents the matching of the decision to its transmission unit and the decision branching and enforcement depending on the result. If the result is PASS, the transmission unit is forwarded to its destination; if the result is FAIL, an exception occurs (represented by the solid triangle in the diamond), and the transmission unit is dealt with accordingly (e.g., recorded to the audit log, and then discarded). The separation of SF into two boxes serves to further illustrate the bi-directionality of communications.

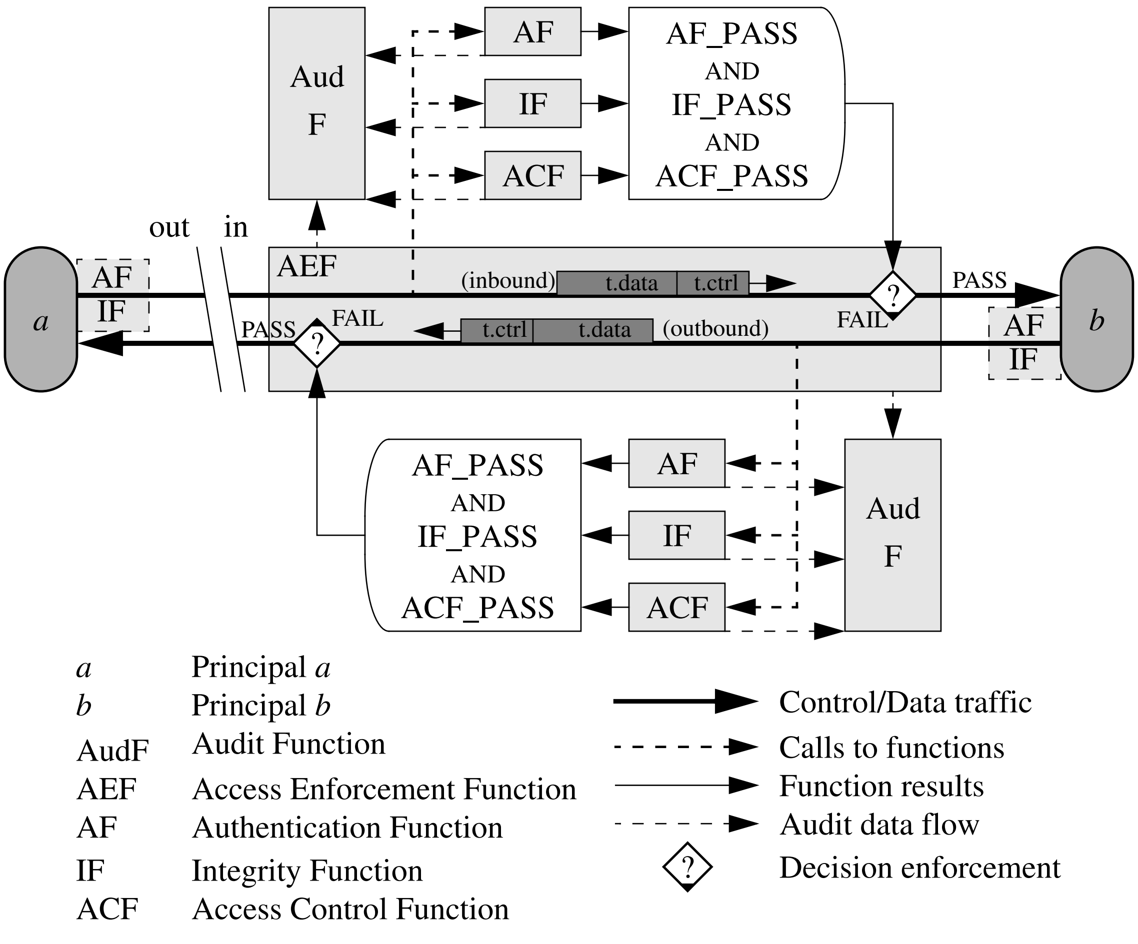

Figure 5.2 Reference model for firewall technology. The model displays a number of security functions that are applied to communication traffic entering and leaving a network policy domain, capable of enforcing (AEF) the security services of *authentication* (AF), *integrity* control (IF), *access control* (ACF), and *audit* (AudF), or a subset thereof. The model is applied at a protocol layer or a range of protocol layers.

Figure 5.2 depicts a more detailed representation of the reference model of firewall technology. It further qualifies the structure of SF, includes an access enforcement function, and illustrates possible participation by sender and destination.

The *access enforcement function* (AEF, see section5.2.5) located in the communication path between these two principals may request the authentication of each transmission unit, the verification of the integrity of each transmission unit, the access control decision, and enforce the results of these functions.

Transmission units may need to be authenticated to assure that their apparent and actual origins are identical (*authentication function*, AF; see section 5.2.1). The integrity of the transmission units can be verified by the *integrity function* (IF; see section 5.2.2).

The *access control function* (ACF; see section 5.2.3) determines if the transmission unit is to be forwarded further into the protected network and toward its destination. This decision can be based on control information in each transmission unit or on data contents in the case of content filtering, such as the search for Java applets or computer viruses.

Arrows with thin, dashed lines indicate possible invocations of the *audit function* (AudF; see section 5.2.4). All blocks that are part of the firewall system have invocation access to the audit function to record events and data according to the network domain security policy in force.

For any network transmission unit, functions AF, IF, and ACF can be called in any order. Their results are considered for the decision if the transmission unit should be forwarded toward its destination. There are firewalls that do not implement all these functions at any level of the protocol stack. Although they cannot meet the complete functionality as present in the model, they may be sufficient to implement a particular network domain security policy. The logic gate symbol ( ) indicates the combination of the results of the three functions into a single PASS/FAIL: if a single function generates FAIL as a result, the transmission unit should not be forwarded to its destination.

Outbound communication traffic is subject to the same security functions as inbound communication traffic. However, because of the trust relationship between the firewall and internal principals, it may not be necessary to enforce the same functions as on inbound traffic. For example, if a "trust relationship" exists between internal hosts and the firewall, a firewall designer may choose to omit outbound authentication verification of communication traffic.

Authenticity verification, integrity verification, and access control decisions may be performed close to the network perimeter of the guarded network policy domain or further inside. In both cases it needs to be assured that any possible path that transmission units can take toward the destination in the guarded network policy domain is protected by these functions. We will discuss this issue further in section 5.3.

The dashed boxes with labels AF and IF close to principals a and b indicate cooperation by a sender for the authentication and integrity functions. Cryptographic protocols, the primary means in network security to provide authentication and integrity assurance services, may require the participation of the sender (e.g., to provide cryptographic secrets for the generation of session keys). Without this cooperation, cryptographic protocols could not be used to provide the necessary services of AF and IF. The box is dashed to indicate there are authentication procedures that do not require participation of the sender, and to represent that the participation is not under control of the firewall.

There are certain constraints on the interaction of the components. For example, before the result of a call to the access control function results in communication traffic being forwarded toward its destination, the authenticity and integrity of the arguments of the function invocation must be assured. Without great confidence in the authenticity of the arguments for the access control function, its result cannot be trusted.[1] For example, in TCP/IP firewall technology, IP packet filters perform their

[1] This does not imply that the authentication function must precede the access control function.

actions based on the IP header fields present in datagrams, none of which are authenticated. This shortcoming has resulted in the exploitation of system vulnerabilities through, for example, SYN flooding ([SKK+97]) or IP address spoofing ([CER95]).

Figure 5.2 displays functions as monolithic boxes; however, such a characteristic is not meant to be implied as an implementation requirement. The representation is kept at a high level of abstraction to concentrate on the information flows and functional dependency of its components. The representation of the model is independent of its implementation.

The model allows for unilateral and mutual authentication by choosing the appropriate authentication functions on inbound and outbound communication traffic.

The application of the model is not restricted to an end-to-end, end-to-intermediate, or intermediate-to-intermediate discussion because there is no limitation on the choice of principals a and b. In particular, they do not need to be communication endpoints on destination hosts but can be on intermediate switches.

5.2 Components of Reference Model

As mentioned in section 5.1, the reference model consists of the following functional components: authentication function (AF), integrity function (IF), access (admission) control function (ACF), audit function (AudF), and access enforcement function (AEF). This section describes these functional components.

5.2.1 Authentication Functions (AF)

Authentication provides assurance of the claimed identity of an entity. Authentication provides corroboration of the identity of a principal, within the context of a communication relationship. A *principal* is an entity having one or more distinguishing identifiers associated with it. Authentication services can be used by entities to verify the purported identities of principals.

A second form of authentication, called *connection authentication*[2] , provides assurance about the authenticity of the sender of data in a connection and the integrity of transmitted data. Integrity assurance is part of connection authentication. Nevertheless, we treat these two services as separate functions.

According to our definition of the term firewall technology, firewall enforcement operates on communication traffic. It is the task of the authentication function to verify the authenticity of communication traffic based on its identifiers and authentication information (such as authentication protocol specific data). The authentication function is a predicate and returns either `AF_PASS` or `AF_FAIL`.

If there is any incoming communication traffic for which no authentication function is performed, attacks, such as address spoofing, become possible (e.g., [CER95, CER96]). Furthermore, access control mechanisms may produce incorrect results if the source identifier of the access request is not authentic. The same arguments apply for outbound communication traffic. Therefore the AF is a necessary component for network access control.

It is necessary that the identifier that is involved in the authentication process be interpretable at any place along the connection establishment where it might be verified. If identifiers have global significance, this requirement is trivially satisfied. However, this property is usually not necessary. If an endpoint cannot be authenticated, or its identifying label cannot be interpreted, its identity is labeled as "unknown." It is the responsibility of the security policy in force to comprehend this case. A policy might allow unauthenticated traffic on its perimeter network (a network added between the protected internal network and the external network; popularly called a *demilitarized zone network* (DMZ)), but not on its internal networks; or it might allow unauthenticated traffic only to reach anonymous network services, such as anonymous ftp. These types of decisions are made by the access control function and are discussed in section 5.2.3.

[2] The term *authenticated signaling* as defined in chapter 1 and used in chapters 7 and 8 addresses connection authentication, i.e., the authenticity and integrity of the signaling messages.

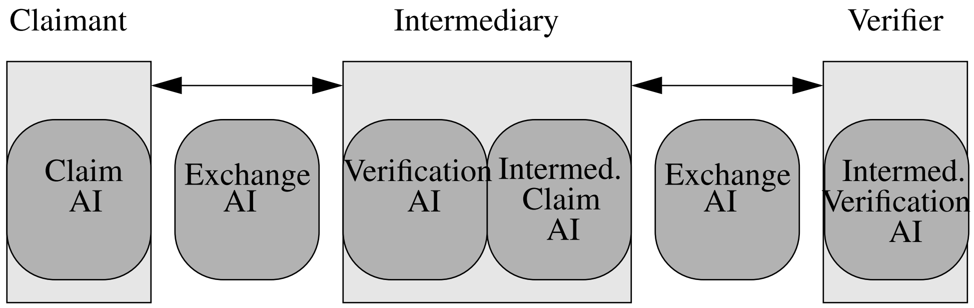

Figure 5.3 Model of in-line[3] authentication. A claimant attempts to be authenticated by a verifier through an intermediate agent. *Authentication information* (AI) is exchanged between claimant and intermediary, and between intermediary and verifier.

The following definitions are consistent with [Ran93a]. Distinguishing identifiers are required for unambiguous identification within a network policy domain. They can be distinguished at a coarse level by virtue of group membership or at the finest degree of granularity identifying exactly one entity. The term *claimant* is used to describe a principal for the purpose of authentication. The authentication *verifier* is an entity which is or represents the entity requiring an authenticated identity. Authentication of a claimant to a verifier is called *unilateral* authentication. An entity involved in *mutual* authentication will assume both claimant and verifier roles.

Authentication methods rely on one or a combination of the following principles: something known (e.g., password), something possessed (e.g., security token), or some immutable characteristic (e.g., biometric identifier) (see [Tuc97, §91.2]).

There are authentication schemes with and without trusted third party involvement (see [Ran93a, figures 1,2]). In the one case no trusted third party is involved. The claimant establishes his identity with the verifier through a direct exchange of authentication information. Third parties can get involved in a variety of ways (see [Ran93a, figures 3,4,5]): in-line[3] (a trusted entity intervenes directly in an authentication exchange between the claimant and the verifier, e.g., ftp proxy; figure 5.3), on-line[3] (one or more trusted parties are actively involved in every instance of an

[3]ITU terminology; see [Ran93a]

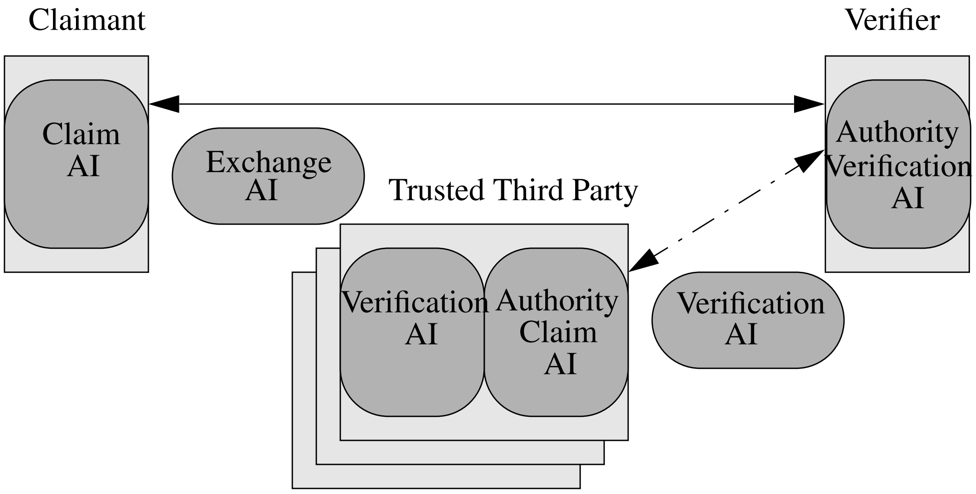

Figure 5.4 Model of off-line[3] authentication. One or more trusted parties support the authentication verification without being involved in each instance of authentication. An example is a verifier using a third party public key repository because public keys are required as part of an authentication protocol.

authentication exchange, e.g., Kerberos; [SNS88]), or off-line[3] (one or more trusted parties support authentication without being involved in each instance of authentication; figure 5.4).

Liebl provides a comprehensive bibliography on authentication in distributed systems in [Lie93]. Notable publications investigating the concept of authentication as a basis for other security services are [BAN89, Nes90, BAN90], [LABW92], [Ran93a], [HA94], and [WL93b].

5.2.2 Integrity Function (IF)

The integrity function protects communication traffic from unnoticed and unauthorized modifications, such as insertion, replacement, or deletion ([Den82, §1.2]). It cannot prevent these violations from happening, but it can detect and flag them after the fact. It is a predicate and returns either `IF_PASS` or `IF_FAIL`.

Connection hijacking, such as the active attack against TCP described in [Jon95], is possible if there is any transmission unit for which the integrity function is not applied. Thus, the integrity function is necessary to protect against network-based active wiretapping.

Although possible, and in cases desirable, to provide an additional data confidentiality service, it is not necessary to assure integrity through encryption of the whole data stream. Integrity and confidentiality services each serve different purposes and have different performance characteristics.

There are a variety of mechanisms to detect modification of data ranging from checksum schemes, such as *cyclic redundancy checks* (CRC), to cryptographically secure digital signatures. Schneier and Stinson describe a number of such mechanisms in [Sch95] and [Sti95]. Keyed MD5 (*message digest 5*; [MS95]) is an example of such a mechanism to provide data communications integrity assurance. We describe the use of some of these mechanisms as part of the integrity function in our prototype implementation in chapter 8.

5.2.3 Access Control Function (ACF)

The purpose of the network access control function is to generate the answer to the question of whether communication traffic is allowed to be forwarded past the firewall toward its destination, or not. This function is a predicate. Its two possible results are `ACF_PASS` and `ACF_FAIL`.

If there is no access control function on incoming communication traffic, access to arbitrary services is possible (e.g., unauthorized file retrievals via the *trivial file transfer protocol* (TFTP); [CER91]. A single TFTP packet is sufficient to form a file transmission request.) Furthermore, without an access control function on incoming communication traffic, data-driven attacks cannot be prevented (e.g., transmission of Java applets containing malicious code; [MSR97]). Thus, an ACF is necessary to provide network access control.

The access control function also needs to be enforced on outgoing transmission units. Otherwise, policies, such as "No access to external Web-sites is allowed during business hours," could not be enforced. A second reason is that this function enables the prevention of information leakage (e.g., an ftp transmission of a password file, or trade secrets.)

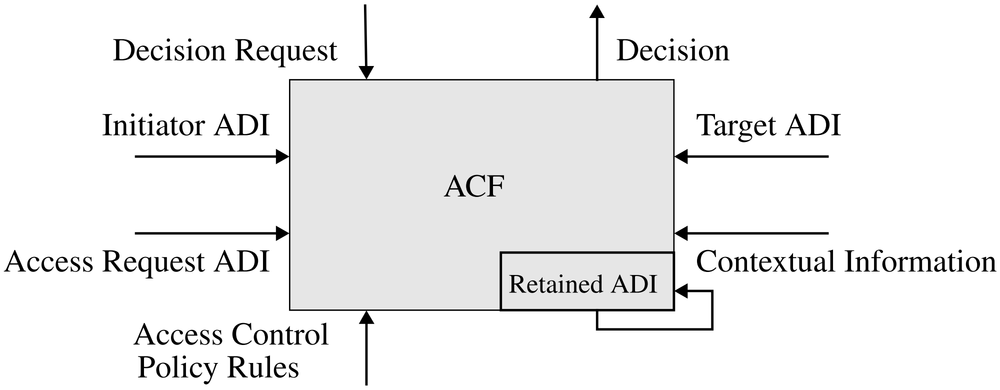

Figure 5.5 Model of the *access control function* (ACF). The ACF serves requests of the form $req(t.ctrl.src, t.ctrl.dst, t.ctrl.acc)\ \forall\ t \in \mathbb{T}$. It takes as inputs a decision request, initiator identifier, target identifier, and access request ADIs (*access control decision information*), an access control policy, and contextual information.

Figure 5.5 illustrates the input/output behavior of a generic access control function. Any such function operates on a subset of the following input information: source and destination information, the type of access request, contextual information, and retained *access control decision information* (ADI). A security policy provides access control policy rules to the decision process. The access control function calculates a result that either allows or denies access, based on the policy and the supplied information.

This model of access control includes two main principals: an initiator and a target. Initiators can be human beings or computer-based entities that access or attempt to access targets. Targets represent computer-based or communications entities to which access is attempted. The access enforcement function is located on any possible path between initiator and target. It is part of the trusted computing base.

The object of the decision, communication traffic $t \in \mathbb{T}$, needs to contribute initiator information ($t.ctrl.src$), target information ($t.ctrl.dst$), and the access request ($t.ctrl.acc$). Source and destination address information are part of the control information, be it conveyed through out-of-band signaling messages (in connection-oriented communications), or as part of packet headers (in connectionless communications). The type of access $t.ctrl.acc$ may not be explicitly present in $t.ctrl$, but encoded through a combination of source and destination address information. For example, in TCP/IP well-known destination port numbers map to services that offer a certain type of access. We assume that external state can be retrieved whenever necessary and that the access control function can retain ADI for later use.

In the case of application-specific proxy servers, the access control function may be part of the proxy service. In this case, some contents of $t.data$ can be input to a fine-grained access control decision. This approach has efficiency drawbacks if applied in a generic fashion at the network perimeter at the network layer. However, the approach can be feasible in specified solutions, e.g., attempting to detect certain types of Web traffic in the data stream to disallow its passing of the firewall. For example, Martin et al. ([MSR97]) explores mechanisms to block possibly hostile external Java applets from passing through a firewall. Such mechanisms can be part of the ACF.

Much research has been performed on the semantics of access control (see e.g., [Den82, Chap.4], [Lun89], [ABLP91], [Ran95], [WL93c], [WL93a], and [YS96]). Several publications propose languages as tools for the specification of access control policies and their enforcement. A rich set of theories and existing implementations can be used.

One such example is *domain type enforcement* (DTE), first described in [BK85]. Domain-based access control takes a hierarchical approach to the scaling issues of access control. It is not feasible to specify security policies exhaustively for all possible participating entities in a large distributed system, be it as *access control lists* (ACL), capabilities, or commpacts [sic] ([RW96]). Domain-based access control represents the structural relationships among entities in a set theoretic approach, e.g., users can belong to a group of engineers, or files can belong to a certain project.

Authentication and access control are inherently related. If we want labels to identify an entity at the highest possible level, these labels can become arbitrarily complex. In general it is not possible for a low-level authentication module in the network layer to perform its operation on this scale because certain high-level information necessary to perform the access control decision is not present at the network layer. This problem is described in [MS91]. It is the conclusion of that paper by Moffet and Sloman that general, application-independent access control at the network layer is not feasible.

It is also impossible to devise a scheme where the access control function at the network layer attempts to negotiate the retrieval and transmission of required information with its peer's higher layers. A paper by Röscheisen and Winograd gives an example that illustrates that the approach of security negotiation in all but the simplest cases becomes a complex synchronization and coordination problem that can lead to deadlock situations ([RW96]). Participants in the negotiation do not know a priori what information the peer requires to make the local access control decision. Including all data that can possibly be needed in the access request is prohibitively expensive and possibly violates privacy concerns of the initiator.

Because of these issues, our model needs to be one of access control delegation. It is the role of the security policy and the firewall system in complex transactions to assure communications occur only with entities (e.g., programs) that are trusted to enforce the policy appropriately. For example, anonymous file transfer requests should be served only by a server whose file system security is known to be appropriate for anonymous file transfer service.

5.2.4 Audit Function (AudF)

The audit function provides the capability of recording an uninterrupted, ordered journal of *significant* system events: what is designated as significant is determined by the security policy in force.

All components of a firewall system need the opportunity to record information in a consistent manner for use by systems, such as notification utilities, audit trail analysis tools, intrusion detection engines, and billing agents ([Pic87]). The information is also provided to authorized personnel for security and system monitoring.

An audit system should be constructed in such a way that if a system violation occurs, the events leading up to and including that violation are reconstructible ([Tal92]). Shimomura and Spafford demonstrate ([Shi95, Spa88]) how audit information may be used in the aftermath of a system violation for the recovery of its functionality and the investigation of what led to the violation. Furthermore, an audit system might allow for the monitoring of systems prior to a violation. Attempts to violate security may then be noticed and acted upon before a violation occurs ([Tal92]).

Audit does not imply the storage of redundant information beyond what is needed to establish monotonicity. However, to achieve fault tolerance, in particular in adverse situations where portions of audit information are deliberately deleted, the storage of redundant information in several locations is highly desirable: redundancy allows cross-checks for the correctness of information. Detected inconsistencies can be a warning sign of tampering. Recorded data needs to be protected from unauthorized modification, retrieval, and addition. The audit system itself needs to be protected against tampering, or the recorded data cannot be trusted. Audit in distributed systems adds several aspects, such as the problem of chronological synchronization of audit events, consistency of record formats, naming issues, and correlation of events for analysis purposes. They are beyond the scope of this dissertation.

Picciotto argues in [Pic87] that the inclusion of a comprehensive auditing facility is a necessary security enhancement for any system. Security policies place various levels of emphasis on the importance of audit: in some cases the availability of the audit subsystem may not be necessary, in others required. In the latter case there is a functional dependency among all recording clients and the audit system itself, similar to the dependency of security services, such as access control on authentication. In that scenario, if the audit subsystem is not present, the remaining system is not allowed to make progress until the functionality of the audit subsystem is restored.

5.2.5 Access Enforcement Function (AEF)

The access enforcement function needs to enforce that the functions explained above (authentication function, integrity function, etc.) are called if required by the network domain security policy. Otherwise the access enforcement function will not receive the necessary indication at the decision points indicated by the diamonds in figures 5.1 and 5.2 to make and to enforce its decision. The logic gate symbol ( ) illustrates that all of the results of the applied functions are part of the decision if the packet is to be forwarded toward its destination (inbound or outbound), or if it is to be discarded.

It is not sufficient to calculate the results of the functions; they need to be enforced. Therefore, without the access enforcement function all of the above attacks (and many more) are possible because of the lack of a guarantee that the results of the functions were enforced.

5.3 Distributed Enforcement

Figure 5.6 illustrates the classical view that firewalls are security devices that enforce security policy close to the network perimeter. The box labeled “Firewall” can take on various configurations, but their common characteristic is the existence of a single “choke point” or a small set of such choke points at the network perimeter ([CZ95, §6]). The firewall is a central point of failure and becomes a performance bottleneck in the presence of high performance networking technologies ([Lyl94]).

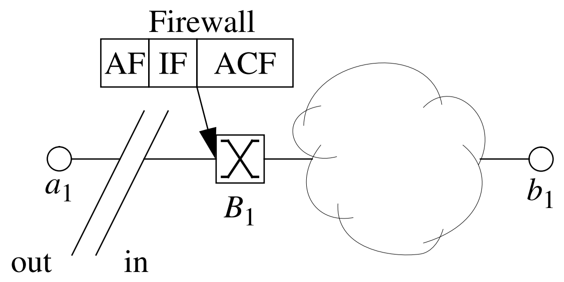

Figure 5.6 Example of classical approach to firewall technology: central application of a focused security enforcement device.

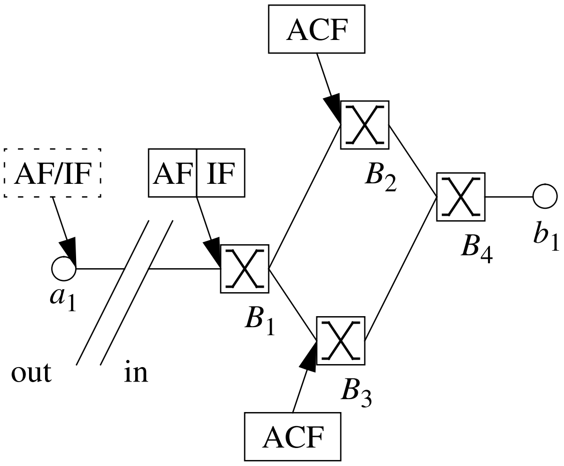

Figure 5.7 Example of distribution of functional components within one layer

As we explain subsequently and validate through experiences with the prototype system described in chapter 8, the distribution of functions offers performance benefits. The firewall functions do not all have to be provided at the same location. They can be distributed. Their distribution reduces the performance overhead experienced at the network perimeter because fewer functions need to be computed there. Functions provided further inside the network can be executed concurrently, thus potentially contributing to an overall performance increase of the firewall with distributed functions. In this fashion, firewall security services can be constructed in an architecture that scales better than previous designs. The distribution of the components may be driven not only by criteria, such as performance increase through replication of functions, but also by the goal to improve reliability, availability, and disaster protection through redundant distribution of functions. Single points of failure can be avoided by design.

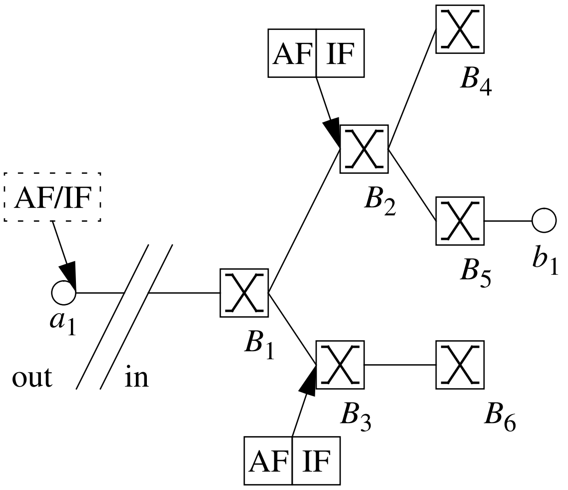

Figure 5.8 Second Example of distribution of functional components

Figures 5.7 and 5.8 depict two examples of the distribution of functions. The example in figure 5.7 is a system in which the authentication and integrity functions are enforced at the perimeter switch, and the access control function (ACF) is replicated across several switches within the network. In such a scenario the ACF must be enforced on any possible path between the sender a_1 and the destination b_1, i.e., on path $P_1 := B_1 - B_2 - B_4 - b_1$ and on path $P_2 := B_1 - B_3 - B_4 - b_1$. If it was not, the ACF could be bypassed, and attacks as described in section 5.2.3 became possible. The example in figure 5.8 locates the enforcement of functions AF and IF further inside the network than the example in figure 5.7.

The following analysis illustrates the possible performance and therefore scaling benefits achievable by a distribution of functions. We characterize the term *time* *overhead* as follows: given a computational platform with certain performance characteristics and given a function that requires computation on that platform, the *time overhead* o is the time required for the function to be computed. Time overheads are always positive.

Definition 5.1 Define o_t^S as the sum of the time overheads that are introduced for a transmission unit t by security functions $S \subseteq \{AF, IF, ACF, AEF, AudF\}$.

Remark. If $S \neq \emptyset$ then $o_t^S>0$ because time overheads are positive values.

Example 5.1 $o_t^{\{AF,IF\}}$ is the time overhead introduced on transmission unit t by both an authentication function and an integrity function. □

We assume that $o_t^{\{AEF\}} \ll \min\{o_t^{\{AF\}}, o_t^{\{IF\}}, o_t^{\{ACF\}}\}$. This assumption is reasonable because the overhead introduced by the AEF consists only of a call to other functions (e.g., AF or ACF) and the enforcement of the result. No high computational effort is required compared to the functions AF, IF, and ACF. Under this assumption we do not consider $o_t^{\{AEF\}}$ any further in the following analysis. We exclude $o_t^{\{AudF\}}$ based on the same argument.

Definition 5.2 Define $O_T^S := \sum_{t \in T} o_t^S$, where $S \subseteq \{AF, IF, ACF, AEF, AudF\}$ and multiset $T \subseteq \mathbb{T}$. We denote O_T^S at a location L as $O_{T,L}^S$.

Remark. $T \subseteq \mathbb{T}$ is a multiset containing elements of $\mathbb{T}$ as defined in chapter 2. T needs to be a multiset because of the possibility of duplicate messages.

Example 5.2 $O_{T,B_2}^{\{ACF\}}$ is equal to $\sum_{t \in T} o_t^{\{ACF\}}$ for all transmission units $t \in T$ at switch B_2. □

For a given $O_{T,L}^S$ in a time interval I the performance characteristics at L may be such that

1. the throughput[4] at L is at or above the throughput of the connected networks, or

[4] including the time overhead imposed by the calculation of the functions in S

2. L operates at a lower throughput than the maximum throughput of the connected networks. In other words, a performance bottleneck exists at L.

Interval I is the time interval between the arrival of the earliest $t' \in T$ at L and the completion of the processing of the $t'' \in T$ that arrived last at L.

In case 2, a reduction of $O^S_{T,L}$ may improve the throughput of the functions in S to such an extent that they qualify for case 1, which is an elimination of a performance bottleneck at L. The following two examples of inbound traffic illustrate the capability of distribution of functions to improve performance characteristics and avoid bottlenecks by design. Both examples assume the computational platforms at B_i for $i \in \{1, 2, 3\}$ are identical.

Example 5.3 In figure 5.6 the overhead at B_1 for a nonempty multiset of transmission units $T \subseteq \mathbb{T}$ is $O^{\{AF,IF,ACF\}}_{T,B_1}$. The overheads in figure 5.7 for $T \subseteq \mathbb{T}$ at B_1 is $O^{\{AF,IF\}}_{T,B_1}$. Comparing the centralized and the distributed design, it is apparent that the overhead at B_1 is reduced from $O^{\{AF,IF,ACF\}}_{T,B_1}$ to $O^{\{AF,IF\}}_{T,B_1}$. Their difference is $O^{\{AF,IF,ACF\}}_{T,B_1} - O^{\{AF,IF\}}_{T,B_1} = O^{\{ACF\}}_{T,B_1}$, where $O^{\{ACF\}}_{T,B_1} = \sum_{t \in T} o^{\{ACF\}}_t > 0$ per definitions 5.1 and 5.2. Therefore this distribution is an improvement. □

Example 5.4 Figure 5.8 depicts a scenario where no firewall function is provided at switch B_1. AF and IF are enforced at both B_2 and B_3 instead. Comparing the distribution of functions in figure 5.7 and figure 5.8[5], and given the assumptions that all transmission units present at B_1 are routed through either B_2 or B_3 (i.e., $T_{B_2} \cup T_{B_3} = T_{B_1}$, $T_{B_2} \cap T_{B_3} = \emptyset$) and not all traffic at B_1 is routed through only one of the switches B_2 or B_3 (i.e., $T_{B_1} \neq T_{B_2}$, and $T_{B_1} \neq T_{B_3}$), it holds that

1. $O^{\{AF,IF\}}_{T,B_2} < O^{\{AF,IF\}}_{T,B_1}$ and
2. $O^{\{AF,IF\}}_{T,B_3} < O^{\{AF,IF\}}_{T,B_1}$.

That is because $O^{\{AF,IF\}}_{T,B_1} = O^{\{AF,IF\}}_{T,B_2} + O^{\{AF,IF\}}_{T,B_3}$ and $O^{\{AF,IF\}}_{T,B_i} > 0$ for $i \in \{2, 3\}$.

[5] Note that $O^{\{AF,IF\}}_{T,B_1}$ refers to figure 5.7. $O^{\{AF,IF\}}_{T,B_2}$ and $O^{\{AF,IF\}}_{T,B_3}$ refer to figure 5.8.

Thus, it is a performance improvement to replicate functions and move them away from the network perimeter as in the distribution in figure 5.8 over the configuration in figure 5.6.

For some methods of providing multicast service, the assumption $T_{B_2} \cap T_{B_3} = \emptyset$ does not hold because transmission units are replicated in the network. In the multicast scenario where some or all transmission units need to be forwarded to both switches B_2 and B_3, the distribution of functions does not yield the full performance benefit as described above. $\square$

5.4 Example Application of Reference Model

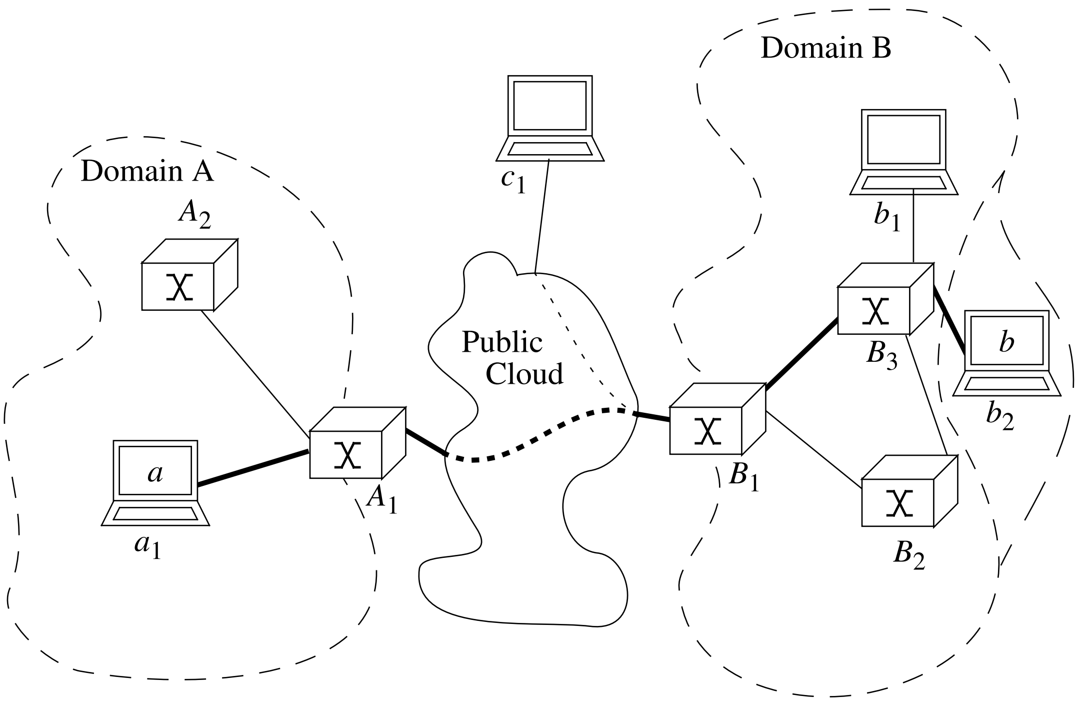

Figure 5.9 Example application of reference model of firewall technology in a connection-oriented communications paradigm.

Figure 5.9 serves as an example of a scenario in which principal a_1 wishes to establish a connection to principal b_1, located in network policy domain B, protected by a firewall.

A number of TCP/IP firewall configurations for this scenario are explained in great detail in [CZ95, Chap.4] and [BC94, Chap.4]. They are based on packet filtering, proxy forwarding, or a combination thereof. In this example we assume a connection-oriented communication paradigm in which a connection between the endpoints must be fully established before data can be transmitted. ATM and TCP can serve as examples of connection-oriented communication technologies.

- Originating principal a_1 initiates a connection to destination principal b_1. Principal a_1 and b_1 are located on opposite sides of the network perimeter that is being protected. a_1 and b_1 can be any of a large set of principals, such as hosts, network interfaces, processes, users, etc.

- As part of the connection attempt, the originator creates at host a_1 credentials for the authenticated call setup (dashed AF/IF in figure 5.2).

- After the connection request arrives at the destination's network boundary (B_1), an authentication module verifies the authenticity (AF in figure 5.2) and integrity (IF in figure 5.2) of the connection request. If either fails, the connection establishment attempt may be terminated at this point, enforced by the access enforcement function. The authentication function logs the authentication request and its outcome (AudF in figure 5.2).

- The call admission control function (ACF in figure 5.2) calculates its access control decision. This function is invoked after the result of the authentication function is known and before the connection is fully established. The access might be refused at this point because of restrictions by a security policy and the connection torn down. Again, the access control decision function logs the decision request and its outcome.

- A positive access control decision might call for further action, such as the validation of the functionality of an enforcement module at destination b_1, or the exchange of enforcement parameters with it. Connection establishment can proceed.
- Once the call is established, a_1 and b_1 can communicate. Further authentication and integrity services can then be used on an end-to-end basis to provide assurance about the authenticity and integrity of data traffic sent over the established connection.

This example illustrates functions to implement network access control, how they are enforced, some of their dependencies, and how their composition creates a system that provides a network access control service. In this example, only the connection establishment is protected. It is not illustrated how the model may be applied again at a higher layer, or how the functions may be distributed. Chapters 7 and 8 serve as a further example of the use of the reference model.

5.5 Scope of Reference Model

The reference model operates on a single network policy domain with its associated network domain security policy. It does not need to address the interaction or coordination of several firewalls operating on disjoint network policy domains: if, for example, an organization decides to use firewalls to enforce the division of its internal network into several network policy domains, it can apply this model separately for each individual domain.

The reference model does not impose a specific approach to the identification of communication traffic; rather it requires external naming, addressing, and directory mechanisms that may be used for name translation. The implementation of mechanisms is not addressed by the model.

A system described by the reference model collaborates with other services outside the scope of the model, such as connection management, data forwarding (or switching) agents, and user processes. It may share state with these services. Although the reference model provides for security services, such as authentication, their implementations are provided externally. The overall system depends on the availability of these services.

We therefore assume the existence of a naming service and a secure key distribution infrastructure (public or private key distribution, depending on the requirements of the used security protocols). Furthermore, we assume the binding between the identifiers of communicating principals and their associated keys are not compromised. The integrity of the trusted computing base and the appropriate strength of used cryptographic algorithms and parameters must be assured.

The implementation of functions by mechanisms can be made independently of how the function is to be used or how the supporting mechanisms are provided.

As defined in our definition of firewall technology, this model operates on communication traffic entering or leaving a network policy domain. It therefore does not address the security problems associated with communication traffic that does not cross a domain's perimeter, as is the case when insiders of an organization launch network-based attacks against their own organization. Some mechanisms such as described in section 3.1.5, however, are capable of protecting against such threats. Nevertheless, this scenario is not addressed as part of firewall technology.

It is necessary to be precise about the perimeter of the network policy domain because a circumvention of a firewall defeats its purpose. For example, it is not obvious if a company-owned laptop, used by an employee on a business trip, is to be considered part of the company's protected network policy domain. For the purpose of our work we assume that the network perimeter is specified in some form, so that it is clear what equipment is "inside" and what equipment is "outside" of the network policy domain, and therefore which communication traffic crosses the perimeter and becomes subject to firewall controls.

As mentioned in chapter 1 the security services provided by a firewall system are only a subset of those required to make a system "secure." Firewalls need to interact with other security aware systems and components. For example, the firewall may allow an anonymous connection to be established to an information server and delegate the file access control decision to that server. The enforcement of access control needs to be synchronized and trusted for these types of delegation.

5.6 Repeated Application of Reference Model

Security services as covered by the reference model at a given protocol layer or range of layers can be expressed by assertions. For example, authenticated signaling in the *asynchronous transfer mode* (ATM), as described in chapter 8, offers endpoint authentication and integrity assurance of signaling messages. Thus, an assertion of the reference model applied to ATM connection establishment traffic is of the form: "For all established ATM connections, participating principals are authenticated and the authenticity and integrity of all signaling messages is verified." This assertion can then be used as a basis for further application of the reference model to communication traffic at other protocol layers.

In designs that include the repeated application of the reference model such assertions are used to determine if the assumptions of security mechanisms at higher layers are met. Through the matching of assertions and assumptions, designers can combine the repeated application of the reference model into a multilayered firewall security architecture.

This approach is flexible and an improvement over the previous state of the technology that favored monolithic designs because it allows the composition of network security mechanisms in layered communication systems. The assertions provide an understanding of which security services are provided by lower layers and can be relied upon. If mechanisms in lower layers are changed, their assertions are likely to change, and designers can determine if the assumptions of higher layer mechanisms are still fulfilled by the lower layers' assertions.

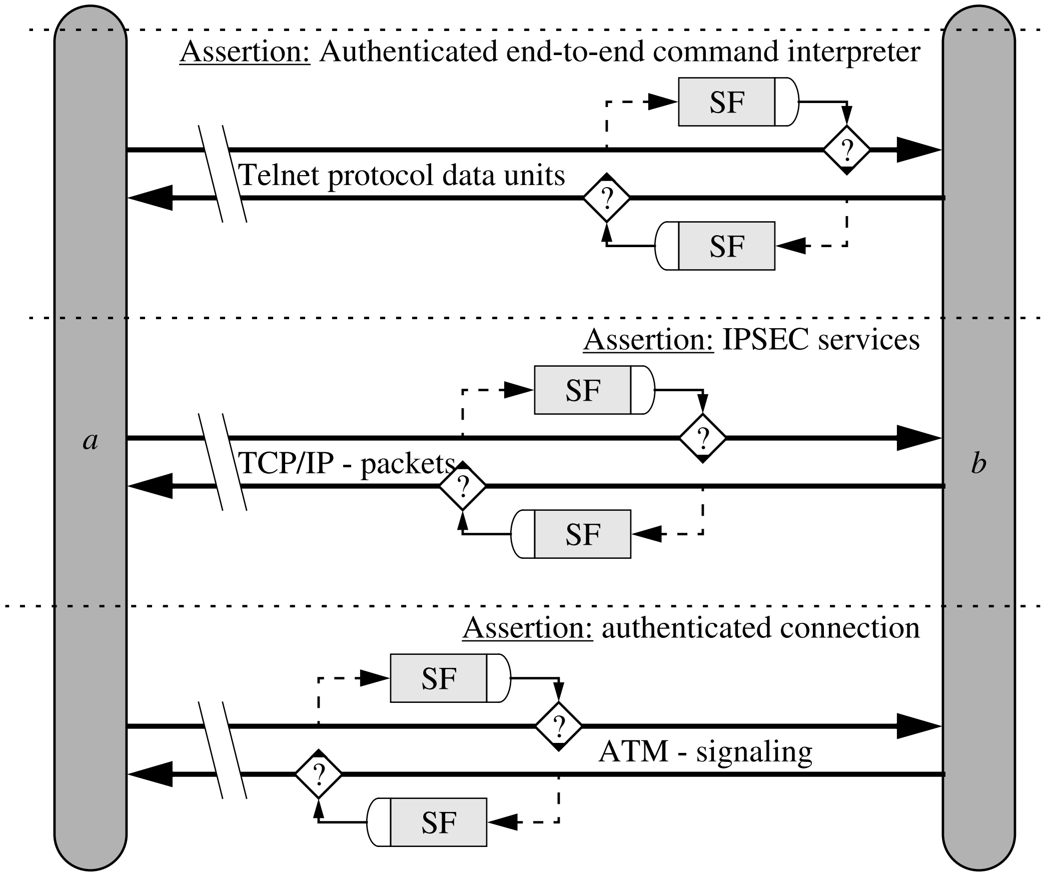

Figure 5.10 Example of the application of the reference model at several layers. It contains an application of security functions at the ATM layer, at the TCP/IP layer, and at the application layer. Each layer offers an assertion about the provided service to the higher layers.

Figure 5.10 illustrates an example of the repeated application of the reference model at several protocol layers: at the ATM layer the application of the model asserts the authenticity and integrity of ATM signaling. These authenticated connections are then used for the transmission of TCP/IP packets. If the IPSEC (*IP security working group*) security enhancements for IP are used, the model asserts at this layer that the integrity and authenticity of each IP packet within the authenticated ATM connections are protected by the IPSEC *authentication header* (AH). Finally, at the highest layer in this example, application layer services (e.g., telnet) can provide additional security services, such as the authentication of the remote user through a password exchange.

5.7 Chapter Summary

This chapter presented the reference model for firewall technology. The functional model is a guide to structure firewall security services at a single layer or a range of layers in a layered model of computer networking. It identifies a fundamental set of functions to providing network access control: authentication function, integrity function, access control function, audit function, and access enforcement function. A characterization of each function is given without the provision of details of how the functionality is to be achieved. Furthermore, this chapter explains how the distribution of the functional components can be used to decrease the performance overhead introduced by firewalls in classical firewall architectures.

6. FORMALISM FOR FIREWALL MECHANISMS AND FIREWALL SYSTEMS

The reference model described in chapter 5 gives a view of the functionality required by a firewall system. Each of its functional components can be implemented by one or more mechanisms. For example, in TCP/IP (*transmission control protocol, internet protocol*) firewall technology both packet filtering and network address translation contribute to the functionality of network access control. Alternatively, a single mechanism may contribute to several functional components, such as an application-specific proxy that performs user authentication and enforces an access control decision. Therefore, there does not need to be a one-to-one relationship between functions (as described in chapter 5) and implementation mechanisms.

This chapter devises a formalism based on *Hierarchical Colored Petri Nets* (HCPN). The goal of this chapter is to develop a practical method, not to develop a logical formalism. Chapter 4 placed this method in the overall picture of the firewall life cycle. HCPNs can be used to express the functionality of mechanisms, to combine them into a system, and to simulate the system in a design tool environment. A number of properties of a modeled system can be analyzed through the application of results in Applied Petri Net Theory.

6.1 Formalism for Firewall Mechanisms: Colored Petri Nets

This section introduces *Colored Petri Nets* (CPN) and *Hierarchical CPNs* (HCPN) as a formalisms for firewall mechanisms and firewall systems. The section commences with arguments why we chose CPNs as a formalism. We then present some limitations of CPNs, introduce the graphical representations of CPNs and HCPNs, and explain their equivalence relationship.

6.1.1 Choice of Formalism: Colored Petri Nets

A *Petri Net* ([Pet62]) is a model expressed as a network of interconnected locations and activities with rules that determine when an activity can occur and specify how its occurrence changes the states of the locations associated with it. Petri Nets have been used for the modeling and analysis of systems ([Pet81]). A considerable body of theory exists ([PR91]) dealing with Petri Net properties, such as liveness and reliability. Petri Nets have been developed over the years from a simple yet universally applicable paradigm to various high-level and more complex but far more convenient methodologies: one such example is Hierarchical Colored Petri Nets. Their formal definition can be found in [Jen91] and [Jen96a].

The following paragraphs are a summary of features that CPNs possess and serve as an overview (see [Jen95, Jen96a] for more details). CPNs promote problem-oriented structuring of a system and make it possible to formulate and prove system characteristics. They offer hierarchical descriptions and are suited for modeling systems of distributed control with multiple processes executing concurrently in time. CPNs are asynchronous in nature without an inherent measure of time although a measure of time has been added in various extensions. Nevertheless, one property of time remains: a partial ordering of the occurrence of events. There are a large number of formal analysis methods by which properties of instances of CPNs can be proved. Computer support for complex analysis methods makes it possible to obtain results that are impractical to achieve manually.

There are other formalisms that are at least equivalent in computational power to CPNs, but not in regard to convenience. Similar arguments apply as in the choice of the best programming language to solve a given problem.

6.1.2 Limitations

The original model of Petri Nets has several limitations that since have been addressed by extensions to the basic model. For example, there is no way in Petri Nets to test for zero occurrence of tokens in an unbounded place ([Pet81]). Although

Petri Nets can be used for modeling systems at various levels of abstraction, in their original form they can be difficult to comprehend by humans, even when a system is expressed at a high-level. Hierarchical Colored Petri Nets are one example of an extended model which already deals with a subset of the original model's limitations.

Many known algorithms that operate on Petri Nets have high computational complexities ([Jen96a, Ch.4,5]). As long as CPNs are only used for their expressive power this limitation is not relevant. But several interesting properties of CPNs, such as boundedness, or the absence of deadlocks, require the application of algorithms with high computational complexity. High computational complexity, however, can still be acceptable if it is sufficient to verify the properties in question infrequently and outside of performance critical paths, such as at the time of design validation ([CLR90, Ch.2]). Because of the high computational complexities for formal verification of properties, high-level formalisms have a disadvantage compared to low-level formalisms, such as the originally defined Petri Nets ([You97]). Section 6.5 gives details on computational complexities and analysis methods for CPNs. In general, low-level formalisms are a better choice for the formal analysis of systems ([You97]). These capabilities are a trade-off for a greater expressiveness in high-level formalisms.

6.1.3 Colored Petri Nets

The CPNs presented in this dissertation use the following notation (cf. figure 6.1). They contain *places* (ellipses) and *transitions* (rectangles). Places (a.k.a. *states*) represent conditions while transitions represent actions. Places can contain instantiations, called *tokens*, of structured data types, called *colors* (italic names next to places), hence the name Colored Petri Nets. The distribution of tokens at places is called a *marking*. The initial marking is determined by an initialization expression (font helvetica expressions next to places; figure 6.2). The marking (boldface font helvetica expression next to places; figure 6.2) for each place is a multi-set (cf. [Jen96a, §2.1] for a definition of multi-sets) over the place's color set.

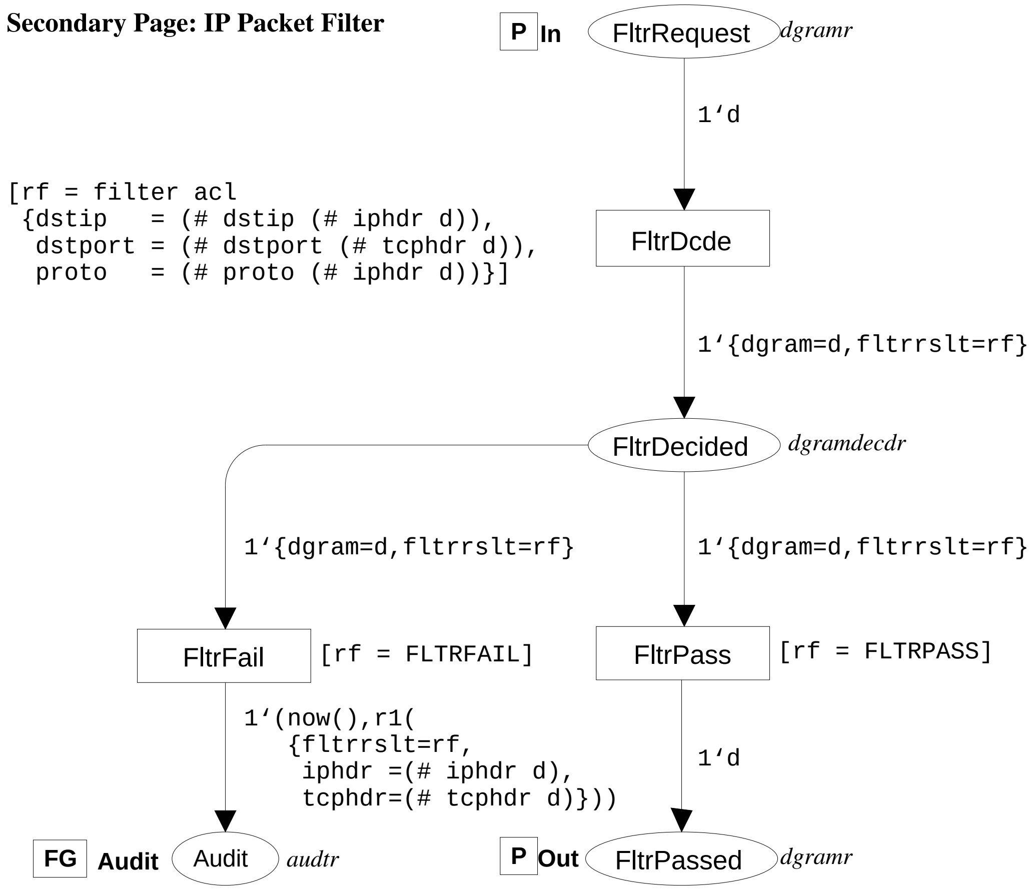

Figure 6.1 Example of a Colored Petri Net for IP packet filtering

Places and transitions can be connected by directed *arcs* (arrows). Transitions are *enabled* if there are tokens in all places associated with incident arcs. An enabled transition can *fire*, if token values are *bound* according to relevant *arc expressions* (typewriter expressions next to arcs) and the *guard* (typewriter expressions enclosed in square brackets next to transitions) associated with the transition evaluates to *true*. A transition fires by removing the required tokens from all places connected through incident edges and by adding tokens to all places connected through emanating edges. Arc and guard expressions may have a set of variables associated with them. The substitution of values for variables can lead to their unification if they have common instances. In CPNs the binding of values to these variables is equivalent to their unification ([Set90, §8.2]).

We use an extension ([Jen96b] and [Met93, Part 3]) of the programming language ML (*Meta Language*; [Mil84, Wik87]) to define colors, arc expressions, guards, and code sections. ML is a strongly typed, high-level functional programming language optimized for abstract data structure specification and manipulation. The strong typing forces designers to be specific about the data types of represented information and ensures unambiguous interface specifications for the combination of CPNs. For the manipulation and simulation of CPNs we use the Design/CPN software from the University of Aarhus ([Met93]). It uses ML as the specification language of choice.

CPNs can be specified formally without a graphical representation: as a tuple consisting of sets (color, place, transition, and arc sets) and functions (node, color, guard, arc expression, and initialization functions), as in [Jen96a, Definition 2.5]. This method of specification of CPNs is necessary for formal analysis methods by which properties of CPNs can be proven. We chose a graphical representation of CPNs over its set theoretic representation to graphically express the structure of modeled systems.

6.1.4 Hierarchical Colored Petri Nets

Figures 6.1 and later 6.4 illustrate separate mechanisms that are used to build firewalls. Firewalls consist of a set of mechanisms that collectively provide network access control. Furthermore, they use external functions, such as authentication header verification, and external state, such as TCP connection state. For practical reasons it is not desirable to create a single large CPN that specifies a given firewall system in a flat structure.

The concept of Hierarchical CPNs allows a designer to construct large CPNs by combining a number of smaller CPNs. They are defined in [HJS91]. HCPNs can be constructed top-down, bottom-up, or by a mixture of these two strategies. HCPNs make it possible to relate a number of individual CPNs to each other in a formal way, and thus allow their formal analysis ([Jen96a]).

In top-down design one starts out with a simple high-level description of a system without consideration of internal details. A specification of detailed behavior of the CPN is developed through stepwise refinement ([Wir71]). Stepwise refinement is achieved through the application of a construct called *substitution transition*, where a more complex CPN takes the place of a transition. The CPN must conform to the interface of the replaced transition and relate identically to its surrounding arcs. Transitions Packet Filter and Authentication Header are examples of substitution transitions.

In bottom-up design CPNs are combined into a larger net through *fusion places*. A fusion place is a set of places that are considered to be identical. Even if they are drawn as individual places they represent a single conceptual place. For each token that is added (removed) at one of the places, an identical token is added (removed) at all others. Places FltrRequest in figure 6.1 and P1 in figure 6.2 are a fusion place, for example.

A non-hierarchical CPN is called a *page*. Figures 6.1, 6.2, and 6.4 contain pages. A page that contains a substitution transition is called a *superpage* (e.g., figure 6.2); a page that contains the detailed description of the activity modeled by the corresponding substitution transition is called *subpage* (e.g., figure 6.1). A substitution transition is also called a *supernode*. Note that the places connected to a substitution transition by a single arc (called *socket nodes*) and their counterparts on the subpage (called *port nodes*) are fusion places. The interface between a superpage and a subpage is defined through *port assignments* where socket nodes are related to port nodes.

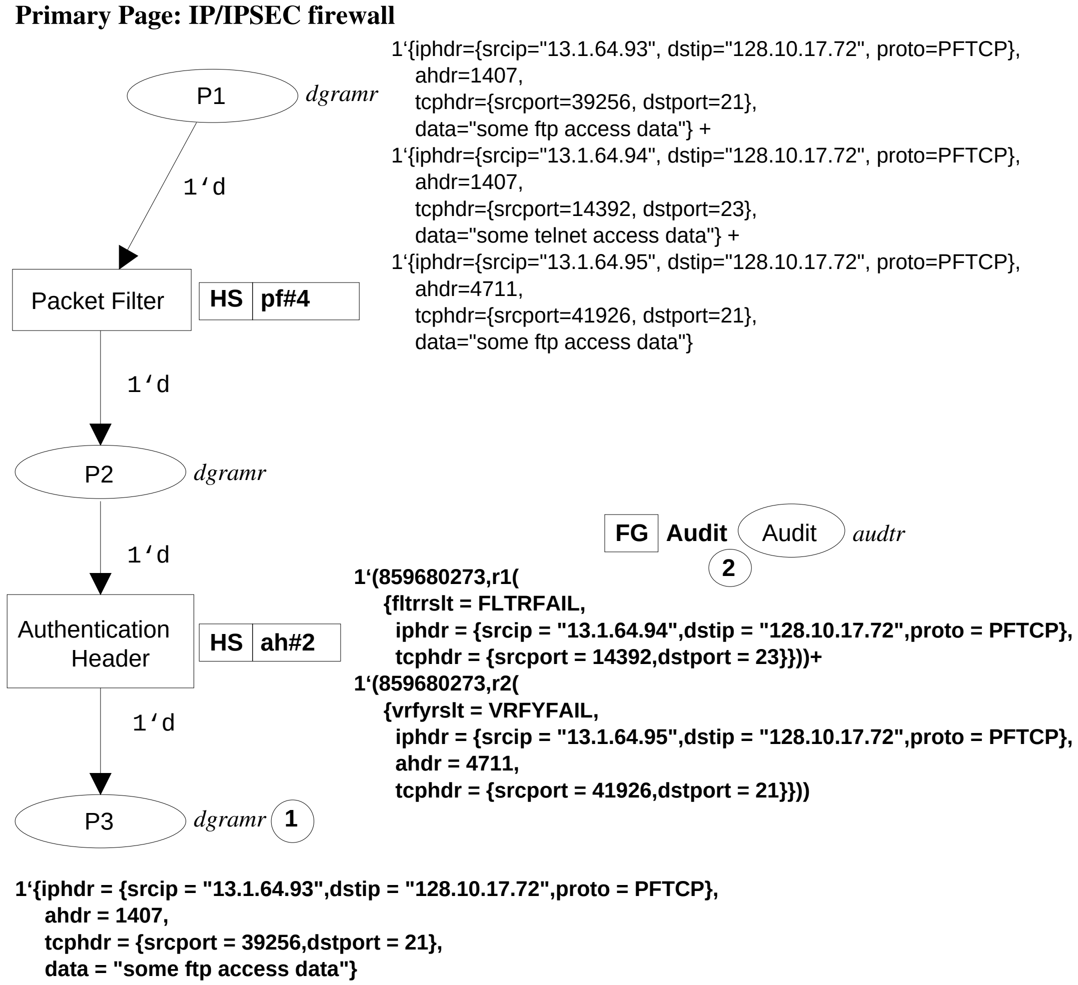

Figure 6.2 Hierarchical Colored Petri Net for a simple IP firewall consisting of an IP packet filter and IPSEC authentication header module.

6.1.5 Equivalence of CPNs and HCPNs

Any HCPN can be translated into a behaviorally equivalent non-hierarchical CPN by replacing each substitution node with a copy of its subpage. This replacement process may need to be applied recursively. The recursion is guaranteed to terminate because a strictly hierarchical relationship between pages is enforced during construction. HCPNs are equivalent to CPNs, which means the theoretical modeling powers of these two classes are identical. However, they have different properties from a practical point of view: HCPNs allow a designer to cope with large systems because of their concepts for structuring and abstraction.

6.2 Example HCPN for a Simple IP Firewall

This section describes an example firewall that combines two firewall mechanisms, an IP packet filter and an IPSEC (*IP security working group*) authentication header module. We structure the description top-down, starting with the superpage.

The firewall system modeled in figure 6.2 is a superpage consisting of two components: an IP packet filter, and an AH (*authentication header*) module. Places P1, P2, and P3 contain tokens of color *dgramr* that represent IP datagrams. The two components are represented as substitution transitions, with the packet filter from figure 6.1 being applied first. Each instantiation d of color *dgramr* in place P1 represents a datagram d that arrives at the firewall. Note that d is a transmission unit as defined in chapter 2 and $d \in \mathbb{T}$. Once substitution transition Packet Filter fires, d is removed from place P1. It is only added to place P2 if d is added to place FltrPassed (figure 6.1), a fusion place of P2, within the subpage.

Thus, only datagrams that pass the transition Packet Filter successfully can be input to the transition Authentication Header representing the IPSEC AH firewall component. All datagrams that are added to place P3 therefore have passed both firewall components successfully and can be forwarded to their destination. Figure 6.2 depicts place Audit which models an audit function collecting audit events.

Remark. The meaning of arcs around substitution transitions, such as Packet Filter, differs from the meaning of arcs around regular transitions. The set of arcs around a substitution transition describes an interface of the substituted CPN rather than a unification of common instances that must occur. It means that datagrams that are removed from place P1 because transition Packet Filter fires need not be added to place P2. They are only added if they appear in the fusion place corresponding to place P2.

6.2.1 IP Packet Filtering

Figure 6.1 gives a CPN specification of a typical IP packet filter as described in section 3.1.1. It models the invocation, filtering decision, and decision enforcement of a packet filter.

Each datagram that arrives at the packet filter is represented by a token of color *dgramr* in place FltrRequest. In this example, a datagram (type *dgramr*) consists of several possible types of headers (types *iphdrr*, *ahdrr*, *tcphdrr*) and a data portion (cf. figure 6.3 for the ML declaration of colors in this example). The header contains a subset of the TCP/IP header fields. It does not contain all header fields as defined in [Pos81a], but rather those that are necessary and sufficient to perform the packet filtering operation in this example. The header fields are used by the packet filter to decide if the datagram is to be forwarded or discarded.

The transition FltrDcde is enabled whenever the marking of place FltrRequest contains at least one token, i.e., whenever a datagram arrives at the packet filter. Variable `d` is then bound to the datagram values, which unifies all occurrences of `d` to this instance. The guard associated with FltrDcde uses function `filter` to apply the access control list defined in `acl` (cf. figure 6.3) against `d` and assigns the result to `rf` (`FLTRFAIL` or `FLTRPASS`). Function `filter` takes two arguments: a list of tuples containing patterns and their corresponding results (`acl`), and a pattern. It returns the result corresponding to the pattern if found in the access control list, and the default safe value `FLTRFAIL` otherwise. The access control policy for IP packets is not encoded in the CPN model, but in `acl`. This CPN model merely describes a mechanism for enforcing whatever policy is encoded.

Once transition FltrDcde is fired, `d` is removed from place FltrRequest. Datagram `d` and its filtering result `rf` are combined into a token of color *dgramdecdr*, a record type, and added to place FltrDecided. Depending on the value of variable `rf` exactly one of the two transitions FltrFail and FltrPass is enabled because both are guarded by mutually exclusive but collectively exhaustive expressions. Note that a guard expression, such as `[rf = FLTRFAIL]`, is no assignment but rather a test for equality: after the

```
color  datat      = string;
color  ipt        = string;
color  portt      = int;
color  protot     = with PFUDP | PFTCP;
color  spit       = int;
color  ait        = int;
color  timet      = int;
color  iphdrr     = record srcip:ipt * dstip:ipt * proto:protot;
color  ahdrr      = spit;
color  tcphdrr    = record srcport:portt * dstport:portt;
color  dgramr     = record iphdr:iphdrr * ahdr:ahdrr * tcphdr:tcphdrr * data:datat;
color  fltrrsltt  = with FLTRFAIL | FLTRPASS;
color  dgramdecdr = record dgram:dgramr * fltrrslt:fltrrsltt;
color  spir       = record spidx:spit   * dstip:ipt * ai:ait;
color  dgramspir  = record dgram:dgramr * spi:spir;
color  vrfyrsltt  = with VRFYFAIL | VRFYPASS;
color  dgramvrfyr = record dgram:dgramr * vrfyrslt:vrfyrsltt;
color  fltrfailar = record fltrrslt:fltrrsltt * iphdr:iphdrr * tcphdr:tcphdrr;
color  vrfyfailar = record vrfyrslt:vrfyrsltt * iphdr:iphdrr * ahdr:ahdrr *
                           tcphdr:tcphdrr;
color  audtu      = union r1:fltrfailar + r2:vrfyfailar;
color  audtr      = product timet * audtu;

(*---------------------------------------------------*)
(*filter = fn : (''a * fltrrsltt) list -> ''a -> fltrrsltt*)
fun filter acl dgram =
  lookup dgram acl handle exlookup => FLTRFAIL;

(*verify = fn : spir -> dgramr -> vrfyrsltt*)
fun verify (s:spir) (d:dgramr) =
  case (# ai s) of
    42 => VRFYPASS |
    _  => VRFYFAIL;

(*now = fn : unit -> int*)
fun now () = tod ();

(*---------------------------------------------------*)
val acl = nil;
val acl = insert {dstip="128.10.17.72", dstport=21, proto=PFTCP} FLTRPASS acl;

(*---------------------------------------------------*)
var d  : dgramr;
var s  : spir;
var rv : vrfyrsltt;
var rf : fltrrsltt;
```

Figure 6.3 Example declaration of colors for the Colored Petri Net model of IP packet filtering and specification of an access control list for IP packet filtering. Access to host 128.10.17.72 is granted for TCP on service port 21 (ftp). All other accesses are denied.

first assignment to variable `rf` in the guard of transition FltrDcde all occurrences of `rf` are unified and assignment and test for equality are denoted by the same symbol (`=`) although they are different operations. Therefore, guards in CPNs are predicates with side effects.

In case transition FltrFail is enabled and consequently fired, figure 6.1 models information about datagram d being added to place Audit. This process can be interpreted as the datagram being discarded and details about the denied access being logged. The place is included to be able to collect information about discarded packets for auditing purposes as well as the validation of the behavior of the packet filter itself. If transition FltrPass is enabled, then d is added to place FltrPassed. Place FltrPassed is the final place in this CPN. Each datagram in a marking of FltrPassed can be forwarded towards its destination.

6.2.2 Modeling the IPSEC Authentication Header Module

This section serves three purposes. It gives a second example of a CPN firewall mechanism (the IP Authentication Header as defined in [Atk95b]), it demonstrates how to build a CPN model for it, and it demonstrates how the model interacts with its environment in an abstract manner (e.g., through use of external state or execution of external functionality).

Section 4 of the IETF (*Internet Engineering Task Force*) standard document for the IP authentication header ([Atk95b]) specifies the procedure a module must perform to verify the authentication header in a received IP packet[1]:

```
''Upon receipt of a packet containing an IP Authentication Header, the
receiver first uses the Destination Address and SPI value to locate
the correct Security Association.  The receiver then independently
verifies that the Authentication Data field and the received data
packet are consistent.  [..]

[..] If the processing of the authentication algorithm
indicates the datagram is valid, then it is accepted.  If the
algorithm determines that the data and the Authentication Header do
not match, then the receiver SHOULD discard the received IP datagram
```

[1] Note: SPI stands for *security parameter index*, an end-to-end security association.

```
as invalid and MUST record the authentication failure in the system
log or audit log.  If such a failure occurs, the recorded log data
MUST include the SPI value, date/time received, clear-text Sending
Address, clear-text Destination Address, and (if it exists) the
clear-text Flow ID.  The log data MAY also include other information
about the failed packet.''
```

This procedure can be divided into 4 steps as follows:

1. Receipt of packet
2. Location of security association
3. Verification of authentication data field
4. Enforcement of authentication verification result

Figure 6.4 depicts the CPN for the AH mechanism. Step 1 is modeled through an instantiation of color *dgramr* in initial place AhRequest. Place SpiDb models external state: the set of established security associations. The lookup (step 2) of a security association is achieved through the matching of the security parameter index field present in the authentication header of datagram `d` against the spi index values in members of the marking of place SpiDb. It is reasonable to model the repository of *security parameter indices* (SPI) in this fashion because SPIs are established by external procedures, such as manual configuration or a key management protocol. Key management mechanisms are used to negotiate parameters other than keys to manage security associations.

The verification of the authentication data field (Step 3) takes place in the guard of transition AhVrfy similar to the way we modeled filtering in figure 6.1. Our model contains a stub routine for the authentication header verification (see function `verify` in figure 6.4). The datagram `d` is then assigned to place AhPassed if the outcome of the verification is positive. Subsequently, the enforcement of the result takes place (step 4).

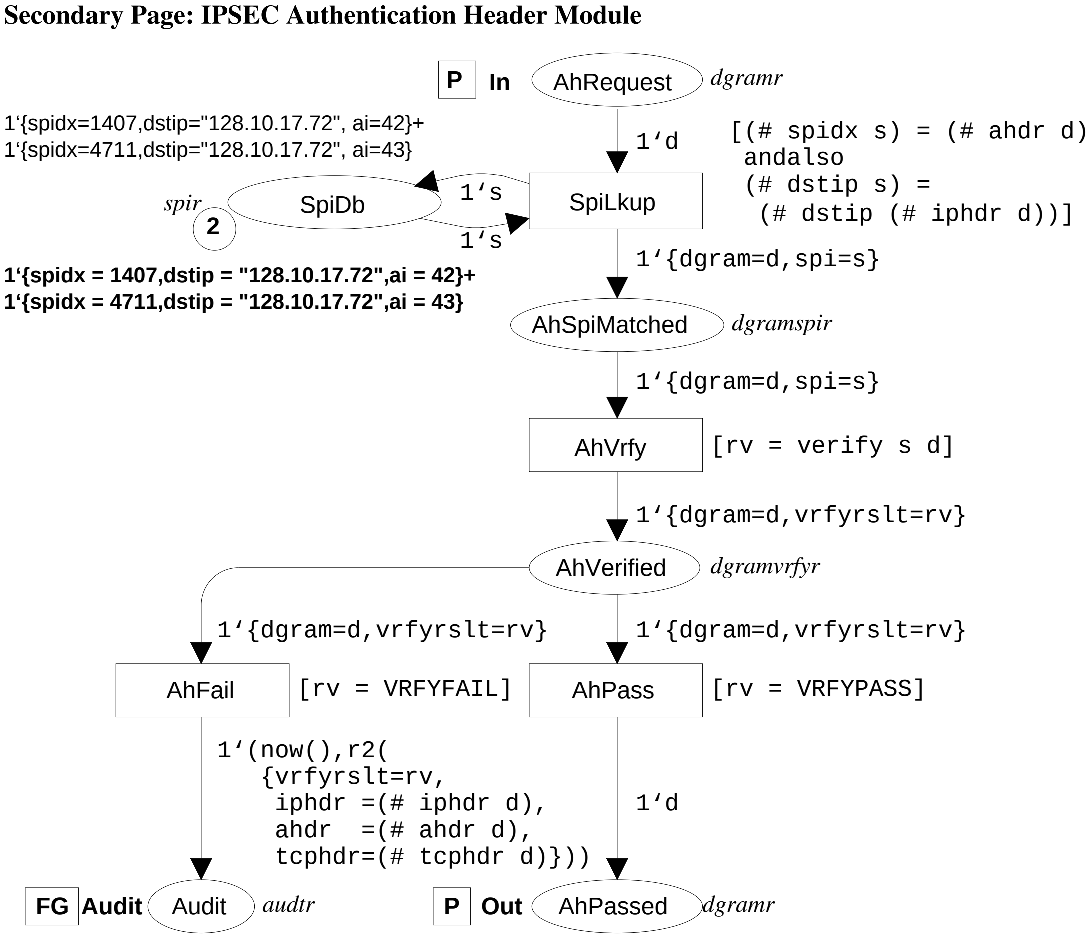

Figure 6.4 Example IPSEC authentication header

In case the result is `VRFYPASS`, `d` will be added to place AhPassed and continue on its path through the system. In case the result is `VRFYFAIL`, certain information from `d` will be augmented by further data fields, such as a time stamp, and added to place Audit. This models the audit requirement as specified in [Atk95b, §4].

The marking of place SpiDb in figure 6.4 contains two examples of security associations. The expression above the place is the initialization expression for the state; the one below is its current marking. The particular values of these markings were used in a simulation and are not of specific interest because they were chosen arbitrarily. Note that in this model the marking of place SpiDb will always be identical to the initial marking because a token that is unified to s for the matching that takes place in the guard for transition SpiLkup is returned to place SpiDb after transition SpiLkup is fired.

6.2.3 Interpretation of Simulation Results for Example Firewall

```
val acl = nil;
val acl = insert {dstip="128.10.17.72", dstport=21, proto=PFTCP} FLTRPASS acl;
```

Figure 6.5 Example access control list. Access is only allowed to the ftp port (21) on the host with IP address 128.10.17.72. (Excerpt from previous figure 6.3.)

We used the designCPN ([Met93]) CPN software for simulations of this firewall model. Figures 6.1, 6.2, and 6.4 display the markings of the model when no more transitions are enabled, i.e., after the end of a simulation. In figure 6.2 place P3 has instances of color *dgramr* in its final marking, and place Audit has instances of color *audtr* in its final marking.

```
(*verify = fn :  spir -> dgramr -> vrfyrsltt*)
fun verify (s:spir) (d:dgramr) =
   case (# ai s) of
     42 => VRFYPASS |
     _ => VRFYFAIL;
```

Figure 6.6 Example verification function implementing the policy that only authentication information 42 is acceptable. More realistically, this function would be replaced by cryptographic authentication code as implemented in our prototype. (Excerpt from previous figure 6.3.)

The tokens that are part of the marking in place Audit recorded events where datagrams did not pass the firewall. The first token contains one audit record describing denied access enforced by the packet filter because the access policy encoded in the access control list `acl` allowed access only to the ftp port 21 (see figure 6.5).

The second token in place Audit represents a datagram that did not pass the authentication verification because we simulated an authentication failure for security parameters index 4711 and its bound authentication information 43. Only authentication information 42 leads to a positive verification result in function `verify` in figure 6.6. Finally, the token in place P3 passed both the packet filtering and the authentication verification and reached the final state of the CPN.

The color of the audit place is a product type containing a time stamp and a union type. The union type depends upon the type of the logged information. Events are logged by adding an arc from the transition representing the event's action to an audit place. In our example we used a fusion place for modeling the audit so that all logged events are collected in the same place. Information present at a transition as well as global state can be logged. Information (in contrast to events) does not need to be logged at the earliest transition because it is represented in a natural way in this model: instantiations of colors. It could be carried through the execution of a CPN as part of a token. We do not postulate which information is logged at what place in the net but we provide a method for modeling audit mechanisms. The research of others addresses the question of content (see e.g., [Pri97]).

The example in section 6.2 and figure 6.2 demonstrates the distribution of functionality as desired in the reference model in section 5.3. The *access control function* (ACF) is provided by the IP packet filter, and the authentication and integrity functions are provided by the IPSEC authentication header module.

6.2.4 Challenges of Modeling

We gave an example of how to model firewall mechanisms in section 6.2.2. Creating CPNs is a task that requires human expertise and experience, similar to other modeling techniques. Jensen provides a number of guidelines in [Jen96a, §1.5] that can help modelers to develop CPNs.

Designers need to understand the behavior of a mechanism before they can formalize it, which represents a problem if a given firewall mechanism is offered as a closed platform only. Designers can infer its behavior only through observation, marketing brochures, and possibly reverse engineering. Possible mismatches between the real firewall mechanism and its functional description can be difficult to detect.

One can imagine that vendors will provide formal descriptions of the behavior of their products as a service to their (potential) customers. Using the tool, guided by the formalism of the HCPNs, and using a library of generic firewalls and specific CPNs for firewall components, we can provide a beneficial design environment. Silva and Valette ([SV89]) argues that catalogs of well tested subnets allow component reusability, which in turn leads to reductions in the modeling effort. The availability of such a library should encourage the adoption of HCPN technology by firewall designers, who would not need to create the models from scratch. This library would enable designers to explore various firewall designs using the available product formalizations in a simulation environment.

6.3 Simulation

Once a firewall mechanism or even a complete firewall system is modeled, our approach allows the simulation of HCPNs. Statistics of simulations can provide insights about characteristics, such as timing constraints and capacity requirements and simulation enables the exploration of various designs.

6.3.1 Testing

Simulation enables testing: for example, recorded sequences of datagrams can be played back as input to a CPN simulator modeling the behavior of a firewall design under consideration. Sequences of datagrams representing attacks could be constructed to determine how a firewall design can handle them.

This approach can be used not only to examine the behavior of the modeled system but also to validate the enforced security policy. The approach is not sufficient to prove correctness of a system, but can at best reveal errors, similar to software testing ([Jen91]). There are a number of different software testing methods. [GJM91, §6.3] gives an overview of software testing and explains its theoretical foundations, principles, and methodologies. Some of the methods used in software testing can be applied here. An exploration of testing enabled through simulation is outside the scope of this dissertation and can be subject to future research.

6.3.2 Performance Analysis

The performance of CPNs can be investigated through the use of *Timed Colored Petri Nets*. Several extensions to CPNs to introduce time are possible. Jensen's Timed CPNs are CPNs where places and transitions *consume* time and tokens are augmented by a time stamp. Time stamps contain the time after which a token is ready to be consumed by a transition. A global clock (discrete or continuous) keeps track of the simulation time. Simulations in timed CPNs are run analogously to discrete event simulations.

Values, such as "the average number of tokens in a given place" or "the average waiting time of a token in a given place," can be determined by simulations in timed CPNs. This approach is useful because analytical solutions through other formal approaches, such as Markov chains, may not be usable because their equation systems become too complex to solve ([Jen95]). The introduction of timing information can result in infinite occurrence graphs (see section 6.5.1) for systems that have finite occurrence graphs otherwise ([Jen95]). By specifying equivalence classes over the time domain one can limit these infinite occurrence graphs to finite subgraphs for which the dynamic properties and performance characteristics can still be determined ([Jen95]).

6.3.3 Design Tool

Exploration of various designs is desirable because critical aspects of systems, such as single points of failure, can be determined. Compared to prototyping, system simulation is a low cost alternative for design exploration. Furthermore, this approach is beneficial over a white paper evaluation because dynamic properties of components can be explored. Note that this approach is not specific to firewall design, but system design in general. McLendon and Vidale describe a similar approach in their research on modeling and analysis of an ADA system in [MV92].

6.4 Automatic Generation of Firewall Code

An approach for "automatic" firewall software development is to integrate design and implementation similar to the integrated software development methodology described by Pinci and Shapiro in [PS91]. Their case study describes the development of a software application for electronic fund transfer. It was built for the Marine Midland Bank of New York and Societé Générale. The system requirement analysis and specification were done using SADT (*Structured Analysis and Design Techniques*); the system design and verification used HCPNs and the implementation was supported with the automatic production of executable SML (*simple ML*; [Jen96b]) code by the HCPN tools.

Applying this approach, there are a number of opportunities for optimization to generate code, which have not been explored. Correctness and efficiency of firewall implementations are of interest because firewalls impose performance overheads for communication traffic on which they operate. In cases where correctness considerations outweigh performance considerations, generated and less efficient, but functionally verified code may be preferable over more efficient code based on ad hoc methodologies.

Brooks expresses scepticism about "automatic" programming ([Bro95, Ch.16]) in all but application domains where problems can be characterized by relatively few parameters, where a library of methods of solution are known, and where analysis has led to explicit rules for selection of solution techniques.

6.5 Analysis of Colored Petri Nets

The early detection of design errors in software engineering saves design time and costs ([Bro95]). Jensen ([Jen96a, Ch.4,5]) lists a number of possible properties of Colored Petri Nets that can be analyzed by informal or formal analysis methods to detect such design errors. The properties are divided into static and dynamic properties. Static (or structural) properties characterize CPNs without consideration of possible occurrence sequences while dynamic (or behavioral) properties characterize the behavior of instantiated CPNs. In general, dynamic properties are more difficult to verify than static properties, especially if one relies on informal methods. Formal analysis methods for dynamic properties can be of high computational complexity because they need to explore large combinatorial spaces.

6.5.1 Occurrence Graphs as the Basis for Analysis

Occurrence graphs are directed acyclic graphs (see [CLR90, §5.4]). Their nodes represent the reachable markings of CPNs, and their arcs represent variable bindings between nodes. Their construction is a partially decidable problem. An algorithm exists that halts if and only if the occurrence graph is finite. Otherwise the algorithm does not terminate ([Jen95, Prop. 1.4]).

The possible state explosion of occurrence graphs is a known problem ([JK91, MV92]). One can apply ad hoc reductions of occurrence graphs. However, those reductions usually do not preserve the behavior and properties of the original model. Jensen discusses in [Jen91, §4.2] a variety of approaches for the reduction of occurrence graphs, such as by means of covering markings, by ignoring some of the occurrence sequences that are identical, by means of symmetries, or by expressing states as symbolic expressions.

Even if the occurrence graph is finite, its construction may still take a long time because occurrence graphs are generally large. The size of the graphs is dependent on several factors, such as the modeled problem or the required color sets and their domains. For example, the number of nodes in the occurrence graph for the dining philosophers problem as modeled in [Jen95, §1.6] grows as a Fibonacci series, i.e., $N(n) = N(n-1) + N(n-2)$, where $N(2) = 3$ and $N(3) = 4$. The growth of Fibonacci numbers is exponential ([CLR90, §2.2]).

The construction of the occurrence graphs is the dominant cost in the analysis of dynamic properties of CPNs. Many algorithms of interest to us, such as those described in [Jen95] that operate on occurrence graphs, are of at most polynomial complexity. Therefore, the smaller the occurrence graph the lower the requirements for computation time. There are methods that reduce the size of occurrence graphs by exploiting symmetry and equivalence relations (see [Jen95, Ch. 2,3]).

6.5.2 Invariants

Consider predicates that may be applied to the states of a system. A predicate is called an invariant if and only if it is valid in each state. Jensen explains the theory and use of invariants in [Jen95, Ch. 4]. In CPNs there are place and transition invariants, and they are applied in the following way: First, equations are formulated that are postulated to be always satisfied. Second, it is proven that the equations are indeed satisfied. Third, the equations are then used to prove some of the dynamic properties of the modeled system (e.g., reachability, boundedness, home, liveness, and fairness). Place invariants are interpreted as sums of tokens that remain constant with the firing of transitions. Transition invariants deal with repetitive firing sequences.

Invariant analysis can prove structural properties of a CPN independent of its marking. Invariants have an advantage over occurrence graphs insofar as they avoid the possible state explosion.

Invariants over the number of datagrams in a net can be used to answer questions about firewall mechanisms, such as these:

- Did all datagrams that reach one of the final acceptance states originate in an authorized start state? A verified invariant to that extent assures that no transmission units that reach final states can get introduced into the firewall through means other than placement in initial states. In other words, firewall controls for accepted packets be bypassed.

- Do certain attributes of transmission units adhere to a desired functional relationship? A first example of such a functional relationship is the identity function. It can be used to ensure that attributes, such as destination machine address and port numbers, do not change during net execution. A second example is a function mapping internal addresses to externally visible addresses, such as in *network address translation* (NAT) firewall mechanisms (see section 3.1.2).

- Do all transmission units reach one of the defined final states representing acceptance or rejection? An invariant to that extent that holds would assure that no transmission units can get lost in the firewall implementation. Such a loss would be interpreted by the outside world as a possibly incorrect rejection of the datagram.

6.5.3 Static Analysis

Jensen defines in [Jen96a, §4.1] a set of static properties on arc expressions and transitions. A static analysis of the type of CPN models that are generated by our approach (e.g., in figures 6.1, 6.2, and 6.4) reveals that all arc expressions are *uniform* with *multiplicity* 1, all transitions are *uniform*, all transitions are *conservative*, all transitions, except transition SpiLkup have the *state machine property*, the primary page net in figure 6.2 is *open* because it has places as border nodes, and all secondary page nets are *closed* because they have transitions as border nodes.

The previous properties determine that the CPNs *as constructed* have a simple structure. Most transitions are conservative with the state machine property because they represent actions on single transmission units (for example datagrams). The transitions output single data items (for example a transmission unit augmented to a compound data structure by a result of the action taken: record *dgramvrfyr* as in transition AhVrfy).

Such a simple structure is preferable over a more complex structure because it adds less complexity to the occurrence graph. As we argued in section 6.5.1, smaller occurrence graphs are an advantage during the formal analysis of dynamic properties.

6.5.4 Dynamic Analysis

The dynamic analysis of CPNs explores properties, such as *boundedness*, *liveness*, *home markings*, *conservation*, *reachability*, *coverability*, *firing sequences*, *equivalence problems*, and *subset problems*. Definitions for these properties can be found in [Pet81] and [Jen96a]. In the following we examine two of these properties in more detail: boundedness and liveness.

Safety properties stipulate some *bad* condition never occurs during the execution of a net. Examples of safety properties are boundedness, reachability, mutual exclusion, and freedom from deadlock.

Bad conditions can be represented by an assertion, P_{bad}, which is mapped to *true* in exactly those states in which the condition is *true*. Therefore, if the safety property is *true* of a net, no occurrence sequence can contain such a state. Hence, the bad condition happens at some particular point in the execution. For a safety property to be *true* of a net, $\neg P_{bad}$ must be a net invariant. One way to demonstrate a safety property is to find a *true* program invariant I, so that $I \Rightarrow \neg P_{bad}$. Another way to express this idea is through the use of temporal logic ([Pnu77]). Temporal logic introduces two temporal operators on assertions: $\Box$ (always) and $\Diamond$ (eventually). A safety property can be expressed as: $\Box\neg P_{bad}$.

The *liveness* property stipulates eventually some *good* condition Q_{good} will occur during the execution of the net: $\Diamond Q_{good}$. Owicki and Lamport ([OL82]) presents a formal proof method based on temporal logic and proof lattices for deriving liveness properties of concurrent programs.

6.5.4.1 Boundedness

Upper (lower) bounds on places indicate the maximum (minimum) number of tokens of a particular color that can be in that place at any given time. The simple firewall model in sections 6.2 has no upper bounds imposed on its places. In particular, place Audit is not bounded.

The CPNs for the packet filter in figure 6.1 and the authentication header module in figure 6.4 are modeling mechanisms that are limited to processing one datagram at a time. This restriction places upper bounds of 1 on all but the initial (FltrRequest and AhRequest) and final places (FltrPassed, AhPassed, and Audit) of these CPNs. Places that represent external state, such as SpiDb, are subject to their own boundedness constraints.

One may need to specify that an event may happen only when a given condition does not hold, i.e., when the corresponding token is absent. The attempt to match a *dgramr* token to a *spir* token in transition SpiLkup in figure 6.4 can serve as an example. In the given model, if for a given token d no matching token is present in place SpiDb, token d cannot make progress towards places Audit or AhPassed. To solve the liveness problem in nets with unbounded places, Heuser and Richter suggest in [HR92] to use complementary places. The initial marking of a complementary place is the whole domain of the key of the token color to match. Tests for boundedness can discover these conditions.

Boundedness constraints are also useful to model limited resources. Bounded places imply finite capacities. Determining finite bounds of places can be used to gather information about resource requirements at those places.

6.5.4.2 Liveness

Liveness in a CPN means that a set of binding elements remains active. Liveness in a firewall representation can be interpreted as: every possible datagram starting out in place P1 will eventually reach a *final state* (representing acceptance or rejection of the datagram). This modeling approach implies that a datagram cannot disappear between its entry to the net and its reaching the final state.

6.6 Chapter Summary

This chapter introduced our formalism for expressing firewall mechanisms as well as their composition. We argued that the formalism of *Hierarchical Colored Petri Nets* (HCPN, short CPN) is suited for modeling a concurrent, distributed system with regulated flows of information, such as a firewall system operating on transmission units.

We applied the formalism to a firewall system consisting of an IP packet filter followed by the IP authentication header module. By doing so we introduced CPN terminology and demonstrated how to model a network security mechanism given only its verbatim specification. We built the model in a modular fashion and demonstrated how the hierarchical concepts of CPNs can be used to combine several mechanisms into a comprehensive firewall system. We learned how to model audit in CPN models. After we developed a basic modeling technique, we used the Design/CPN tool for the incremental building, syntax checking, and simulation of firewall models.

We discussed how the simulation of firewall models can be used for firewall testing, for performance analysis, and as a basis for a design tool exploring design alternatives. We outlined how this formalism may be used for automatic generation of firewall software. Finally, we listed a number of static and dynamic (safety and liveness) properties defined for CPNs that can be interpreted as desirable properties in the problem domain of firewall systems.

7. AUTHENTICATED SIGNALING

In the previous chapters we presented a reference model for firewall technology and a formalism for expressing and analyzing the functionality of mechanisms used to build firewalls. The present chapter describes a concept that is applicable to connection-oriented communication protocols: authenticated signaling. Mechanisms that result from its implementation can be used as a building block in the construction of firewalls. Chapter 8 reports on the design, implementation, and exploration of a realization of this concept for *asynchronous transfer mode* (ATM) signaling.

The concepts and terminology introduced by the reference model in chapter 5 are used in the remainder of this dissertation to describe authenticated signaling and its realization. The reference model guided the design and structure of the prototype described in chapter 8.

The concept of authenticated signaling is an abstraction from a standards contribution by Lyles ([Lyl94]). The original proposal in mid 1994 suggested that authenticated user IDs and higher layer protocol addresses be made part of the requirements for ATM connection signaling.

7.1 Motivation

The mechanisms traditionally used to provide firewall security services (section 3.1) are insufficient in the presence of certain characteristics of high performance networking technologies (section 3.4.1): high performance networking technologies can have stringent quality of services requirements and data rates that prohibit processing of data for security purposes during transmission. Such processing would introduce unacceptable delays and delay variation ([PT97]).

This problem is aggravated by the traditional firewall paradigm that requires security enforcement at, or close to, the network perimeter (section 3.1). Furthermore, there is an expectation that the performance development of mechanisms operating on data in transmission will continue to lag behind that of high performance networking technologies for some time to come (section 3.4.2).

Therefore, the approach in our research is not to attempt to apply known firewall security mechanisms at high speeds (as in [Hug96, SHB95, SBHW95, Chu96, TPB$^+$96, Pie96, PT97]), but rather to rely on changes to network architecture and the development and use of a mechanism that was not previously used for this purpose, authenticated signaling.

7.2 Architectural Design

Figure 7.1 illustrates the architectural design of authenticated signaling. It is based on the idea that connection management protocols can provide the transport service required for security protocol messages to be exchanged. They can be used for at least two purposes:

1. to secure the connection management itself (*signaling security*), and
2. to provide basic security services that can be used by a variety of network components (*security signaling*).

The application of the idea of augmenting a connection management protocol to carry security information is not limited to ATM (for which it was proposed in [Lyl94]). For example, the network working group in the IETF has developed a mechanism at the network layer for providing cryptographic authentication of IPv4 and IPv6 datagrams (*authentication header* (AH); [Atk95b]). Interpreting TCP/IP packets as transmission units (where the combination of IP header and TCP header comprise the control information as defined in chapter 2), the AH addition to the TCP/IP standards offers authenticated signaling functionality.

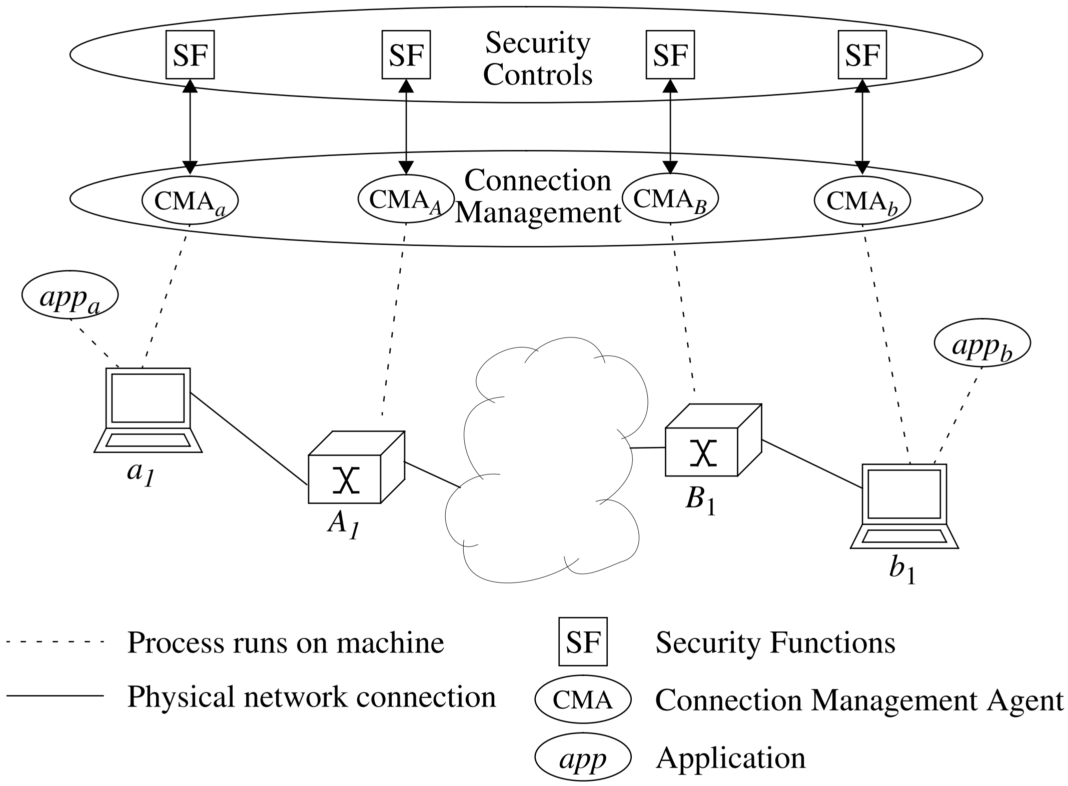

Figure 7.1 Illustration of the architectural design of *authenticated signaling*. A connection management (i.e., signaling) protocol is augmented by security functions (SF) 1.) to protect the signaling messages and 2.) to provide basic security services for other network components. Authenticated signaling is one security control mechanism that achieves both objectives.

7.3 Overview

We use an example to provide an overview of how the design can be used to reach its objectives as given in section 7.2. Figure 7.1 depicts two hosts a_1 and b_1 connected to an internetwork. We assume that a connection is required for applications app_a and app_b running on host a_1 and b_1, respectively, to exchange data. Before application data can be transmitted, the connection management system establishes a connection on behalf of the initiating application (app_a, for instance). *Connection management agents* (CMA) may use services of *security functions* (SF) that participate in security controls.

1. To achieve the first objective, CMAs can use authentication and integrity assurance security services out of SF to sign and verify their own connection management protocol messages.

2. To achieve the second objective, the connection management protocol messages can convey security information on behalf of security protocols processed by functions in SF.

7.4 Discussion

The concept of authenticated signaling has the following distinguishing features:

- *Signaling security* offers the authentication of communication endpoints, the integrity assurance of signaling messages, and protection against denial of service threats (through the opportunity that only authenticated disconnect messages may be accepted and acted upon). As such it can be used to provide the *authentication function* (AF) on endpoint identifiers and the *integrity function* (IF) on signaling message contents as described by the reference model in chapter 5.

- *Security signaling* can provide, for example, authenticated information for network access control decisions, or aid in the provision of data for the generation of non-repudiable audit records.

- The mechanism of authenticated signaling is available to all entities that can interface with the signaling protocol either directly or through intermediate entities. Because higher layer entities can be authenticated by design, they can communicate securely without a duplication of security functions at their layer.

- Authenticated signaling allows the authentication verification at any intermediate connection management agent that has access to the appropriate security function and cryptographic keys. That means the verification is not limited to an end-to-end enforcement.

- The choice of communication endpoints is flexible: any entity that can interface with the signaling protocol can participate in the authentication. This eliminates the need to process special cases for the authentication of entities based on their role (for example "end" vs. "intermediate").

- The following list is a subset of the specifics that the concept of authenticated signaling leaves to the time of its realization in a communication system: choice of connection management protocol, choice of authentication protocol, choice of private vs. public key cryptography, choice of unilateral vs. mutual authentication of entities, choice of clock synchronization requirements, etc.

Some of these items are dependent on others. For example, the set of authentication protocols that can be used is limited by the number of messages exchanged by the chosen connection management protocol. These issues are explored in the following chapter in the context of ATM.

7.5 Chapter Summary

This chapter introduced a concept and its features that can be used to provide firewall security services in the presence of connection-oriented communication protocols. The mechanism is called *authenticated signaling* and can fulfill at least two purposes: it can secure connection management messages (signaling security), and it can be used as a basis for a number of security services, such as network access control or audit.

8. AUTHENTICATED SIGNALING IN ATM

The previous chapter described the motivation, concept, and features of authenticated signaling. This chapter focuses on its realization using the ATM (*asynchronous transfer mode*) signaling protocol Q.2931 and a variety of authentication protocols for experimentation purposes. A system is described that we developed to validate the concept of authenticated signaling and its role in firewall technology through a proof of concept implementation and to expose and explore design issues. The reference model as described in chapter 5 guided the design and structure of the prototype.

The chapter provides some background about ATM and Q.2931 before it describes the software architecture of the prototype. It reports on the exploration of design issues, such as the choice of authentication protocol and integrity assurance mechanisms, issues of cryptographic paradigms, the importance of the number of protocol messages, timing of authentication verification, and clock synchronization.

8.1 Background

There are a variety of connection management protocols (for example, TCP [Pos81c] or Q.2931 [ATM94, chapter 5]). Subsequent sections will describe our research using the terminology from Q.2931. The remainder of this section introduces technologies and terminology required for the understanding of this chapter: the *asynchronous transfer mode* (ATM) (a networking technology that uses Q.2931), connection management concepts, and finally Q.2931. The descriptions in this section are compiled from the ATM *user-network interface* (UNI) Specification ([ATM94]).

8.1.1 The Asynchronous Transfer Mode (ATM)

The asynchronous transfer mode protocols were developed for use in *broadband integrated services digital networks* (B-ISDN) to carry data, voice, images, and video traffic in an integrated manner. ATM is not limited to B-ISDN and contains physical layer and network layer functionality. As of mid 1997 some aspects of ATM are defined by interim standards developed by a user and vendor group known as the ATM Forum [ATM96]. ATM provides for point-to-point or point-to-multipoint, connection-oriented transmission of small data units. ATM gives quality of service guarantees (i.e., it handles resource reservation, offers bandwidth and latency guarantees) to support applications with special service requirements.

Before data can be transferred between machines a connection must be established. These connections are called *virtual channels* (VC) and are identified hop-by-hop using a *virtual path identifier* and *virtual channel identifier* pair (VPI/VCI). Data are transferred in the form of cells with a 48 byte payload and a 5 byte header. Each switch contains mappings of input to output VC identifiers. VCs can be (semi-) permanently installed (called *permanent virtual circuits*, or PVC) or established at connection setup time by a signaling protocol, and released once the connection is no longer required (called *switched virtual circuits*, or SVC). The switching of cells involves the appropriate change of VC information in the cell header and a forwarding of the cell over the associated physical link. This operation can be executed in hardware at high speeds. A switch controller is associated with one or more ATM switches. It performs management functions, such as connection management, enabling or disabling ports, or polling for status information.

8.1.1.1 ATM Protocol Reference Model

A complete description of the ATM Protocol Reference Model is published in [ATM94, chapter 1]. Figure 8.1 depicts an excerpt of the B-ISDN protocol reference model. The planes in the figure are the *user plane* and the *control plane*. The user plane provides for transfer of user application information. It contains the physical

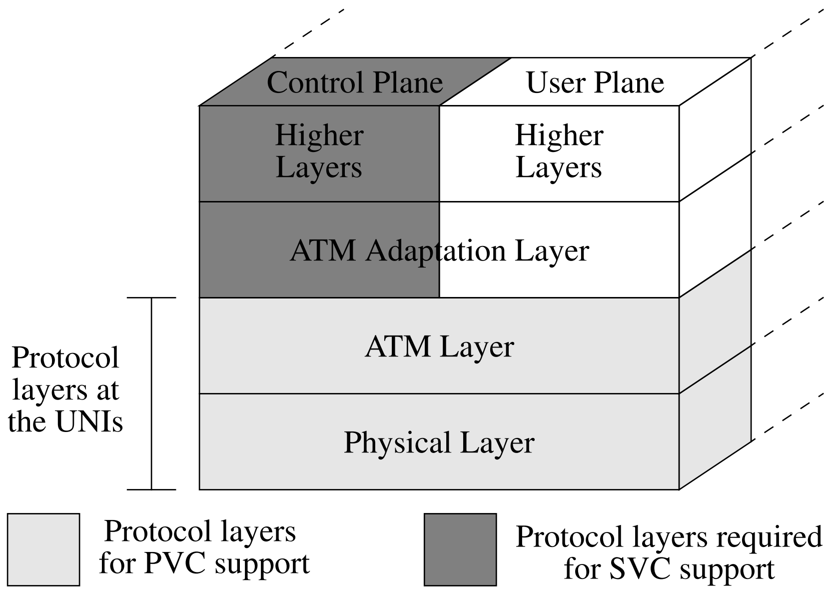

Figure 8.1 Excerpt of the B-ISDN protocol reference model ([For94,figure 1-4])

layer, ATM layer, and various adaptation layers for various traffic types ([Par93, §4.7]). The control plane protocols deal with call establishment, call release, and other connection control functions necessary for providing switched services. It includes the same lower layers as the user plane in addition to higher layer signaling protocols.

Figure 8.2 depicts two types of ATM networks: a private ATM network (managed by the end user's organization) and a public ATM network (operated by network service providers) interconnected by the public *user-network interface* (UNI). Telecommunications carriers within the public network are connected via the *broadband intercarrier interfaces* (B-ICI).

The specification for the public and private UNI is one and the same; the primary distinction between these two classes is physical reach. There are differences between the UNI and PNNI interfaces that are not relevant for our work. Signaling messages exchanged over the UNI and the *private network-to-network interfaces* (PNNI) are identical.

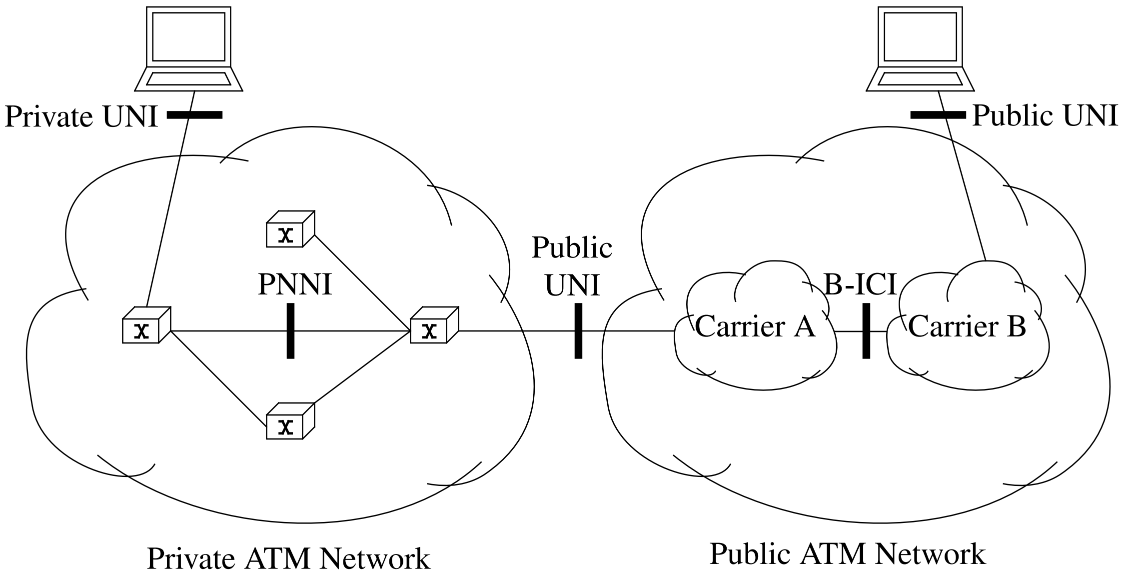

Figure 8.2 ATM interface reference configuration illustrating locations of private and public *user-network interfaces* (UNI), *private network-to-network interfaces* (PNNI), and *broadband intercarrier interfaces* (B-ICI). (Figure courtesy of Tom Tarman, Sandia National Laboratories.)

8.1.2 Connection Management

If communications are based on a connection-oriented model, a connection must be established before user data can be transmitted. We explained in section 8.1.1 that in ATM such connections come in two forms: permanent VCs and signaled VCs. Permanent VCs are statically configured and installed into the switching tables of the ATM switches. They are not subject to connection management. Signaled point-to-point and point-to-multipoint VCs over an ATM network are dynamically established, maintained, and cleared on a need basis utilizing a connection management protocol. The B-ISDN broadband signaling standard ([ATM94]), is the connection management protocol of choice.

At a conceptual level there is a division between the usage of connections and their control and management. Figure 8.3 depicts our illustration of the separation of network switching functions executed in the switching fabric (e.g., the forwarding

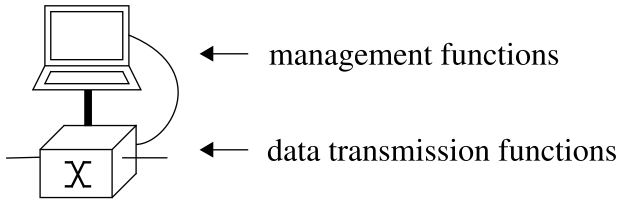

Figure 8.3 Generic switch architecture with separation of management and data transmission functions

of cells over a VC) and management functions executed on the associated processor (e.g., connection establishment and release). This separation is useful for securing the control plane because it enables more general purpose processing power to be assigned to cryptographic computations ([Chu96, §6.1]). Furthermore, it allows switching and control technology to evolve independently.

The initiator of a connection can specify desired characteristics for the connection and relies on Q.2931 to establish or to report the failure of establishment of a call. Q.2931 neither supports any routing functionality nor provides for call acceptance or forwarding policies.

Entities participating in connection management require an associated module to process Q.2931 protocol messages. In figure 7.1 these modules are called *connection management agents* (CMA). In figure 8.4 we further distinguish between *host call control* (*hcc*) and *switch call control* (*scc*) modules (in the following also called processes), which perform signaling functions on the user side and the network side, respectively.

Figure 8.4 depicts the interrelation among the switching processes and their relationship to either ATM switches or end systems. The *hcc* and *scc* processes share state of the local network configuration, such as port identification, routing information, and address information for neighboring *hcc* and *scc* processes. This state can be acquired through static configuration or dynamic discovery and update protocols. The establishment and release of calls is initiated by *hcc* processes. Applications

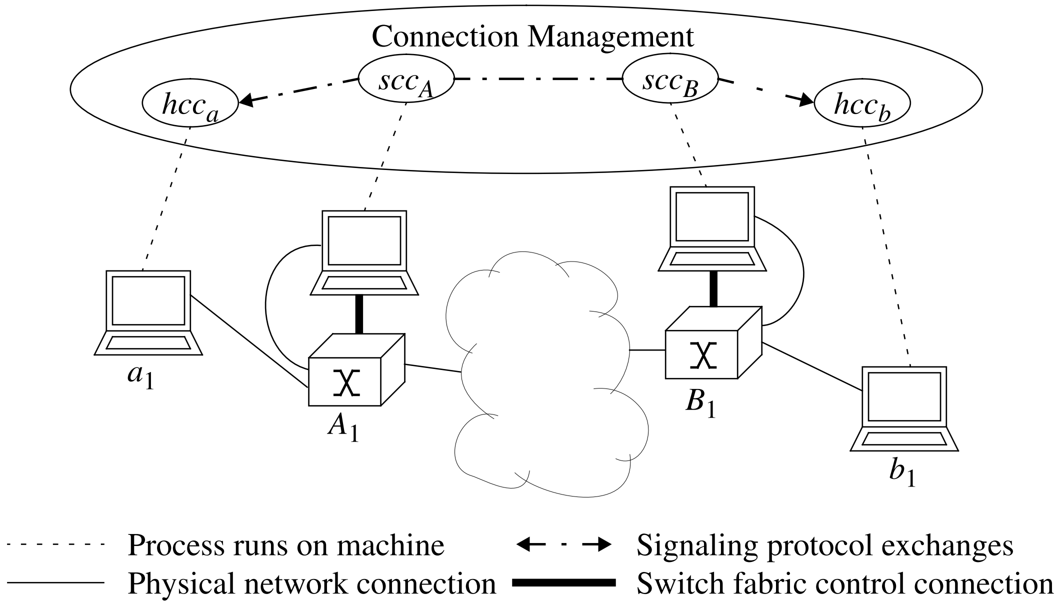

Figure 8.4 Connection management protocols are distributed algorithms which include participation from end stations and network elements.

communicate with their local *hcc* process that in turn communicates with the internetworking sublayer and the Q.2931 module to coordinate their interaction regarding offering, acceptance, and release of calls. Each *hcc* process exchanges protocol messages with its neighboring *scc* process over the user-network-interface. The *scc* processes are responsible for interfacing peer Q.2931 instances on the ATM switches over the network-network-interface. They accept incoming call requests from other *scc* or *hcc* processes and determine the outgoing port to be used when routing a call. Routing information is accessed through a route manager on the ATM switch.

8.1.3 ATM Connection Management: Broadband Signaling Standard Q.2931

Figure 8.5 depicts an example of the Q.2931 message exchange to establish a connection between an initiating sender and a target receiver. Each message consists of a fixed and variable size part. The fixed size portion contains a protocol discriminator, a call reference, the message type, and the message length. The variable size

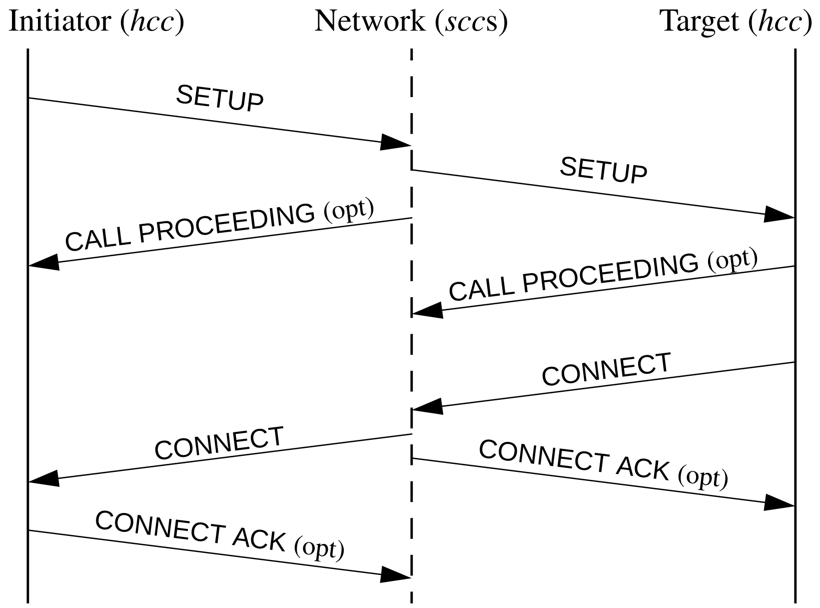

Figure 8.5 Example signaling protocol: broadband signaling standard Q.2931 used by ATM for connection establishment between initiator and target

portion contains a number of *information elements* (IE). There are a variety of IEs, each for a separate purpose. They can contain information, such as an ATM traffic descriptor, ATM adaptation layer parameters, or a called party number. The set of information elements in a message depends on the message type and is defined in [ATM94, section 5.3-4].

The initiating process *hcc* starts the connection establishment by sending a SETUP message. The SETUP message is received by the first *scc* process, optionally acknowledged by sending a CALL PROCEEDING message, and forwarded towards the destination. Although figure 8.5 suggests that the process receiving the SETUP message next is the target *hcc*, there usually are several *scc* processes involved before the SETUP reaches its destination. Upon receipt of the SETUP message the target decides to accept the call or not. In the former case a CONNECT message is sent; in the latter case the call is released by sending a RELEASE message. All *scc* processes involved in a connection establishment process may release or decide not to establish the connection for a variety of reasons. Examples are the unavailability of bearer capability or lack of routing information for the destination of the connection.

8.2 Development Environment

8.2.1 Hardware Environment

The ATM switches we used, called *BADLAN*, were built at the Xerox *Palo Alto Research Center* (PARC). These switches consist of a switching backplane and up to four boards with eight ports each (either TAXI or SONET interfaces). Switch functions, such as the modification of VC tables, are controlled by an external workstation. The configuration has reliability and flexibility advantages. Hosts in our experiments were SUN Sparc stations: IPC, IPX, and SS2. The network adaptors used (called *ParcNic*) were also built at PARC. The experiments were performed over multimode fiber with SONET encoding at OC-3c speeds (155 Mb/s).

8.2.2 Software Environment

The workstations operated under the SunOS 4.1.3 operating system. We implemented a driver for the ParcNic card and used our implementation of Classical IP over ATM for the experiments (see [SKS95] and [SSK98] for a description of our experiences with classical IP and ARP over signaled ATM connections). The implementation of the authenticated signaling required modifications to the signaling code. We used an implementation of the ATM signaling protocol Q.2931 from Bellcore, called Q.port, Release 1.2 ([Bel95]). In its final version the cryptographic protocols were implemented using Bellcore's long integer package, called LIP ([Len95]). The generated cryptographic key pairs were verified with the *Maple* package (University of Waterloo, Canada) and *calc* (Xerox PARC). We generated a set of 60 large (>1000 bits) and 10 smaller asymmetric key pairs and made them accessible to our prototype through a key management database. We investigated further numeric multi precision integer packages from Robert Silverman, called MP ([Sil87]), the DEC Paris Research Laboratories (PRL), INRIA in France, the Information Security Corporation, and the Free Software Foundation (the GNU project). For our prototype LIP was the best overall choice in regard to performance, documentation, licensing terms, and ease of use.

The drivers and authentication and access control services were implemented in C and ANSI C. They are stateless servers that can also be run as daemons in the background. The key benefit of the stateless design was of a practical nature for development and debugging because any service could be halted and restarted without interruption of any instance of the programs that needed to interface with them (e.g., *hcc* or *scc*). The audit function AudF was provided by the UNIX syslog facility. The modifications to the signaling code were done in C++. We used PERL, the Bourne and tcsh shells, and Tcl/Tk as scripting languages for testing and building graphical user interfaces.

8.3 Software Architecture

The following functions are present in the implementation architecture as depicted in figure 8.6. When we speak of *processes* in the following paragraphs we mean the respective functions. We do not mean to imply that they need to be implemented as separate instances of executable programs. We chose to implement these functions as individual programs.

- *app*: Some *application* process using the integrated services network.

- *hcc*: The responsibilities of *host call control* processes are to coordinate the establishment and release of calls between local and remote applications on the host side. Such functionality involves interaction with the application (in our case the ATM network interface card driver) and the signaling protocol to coordinate their activities during the offering, acceptance, and release of calls.

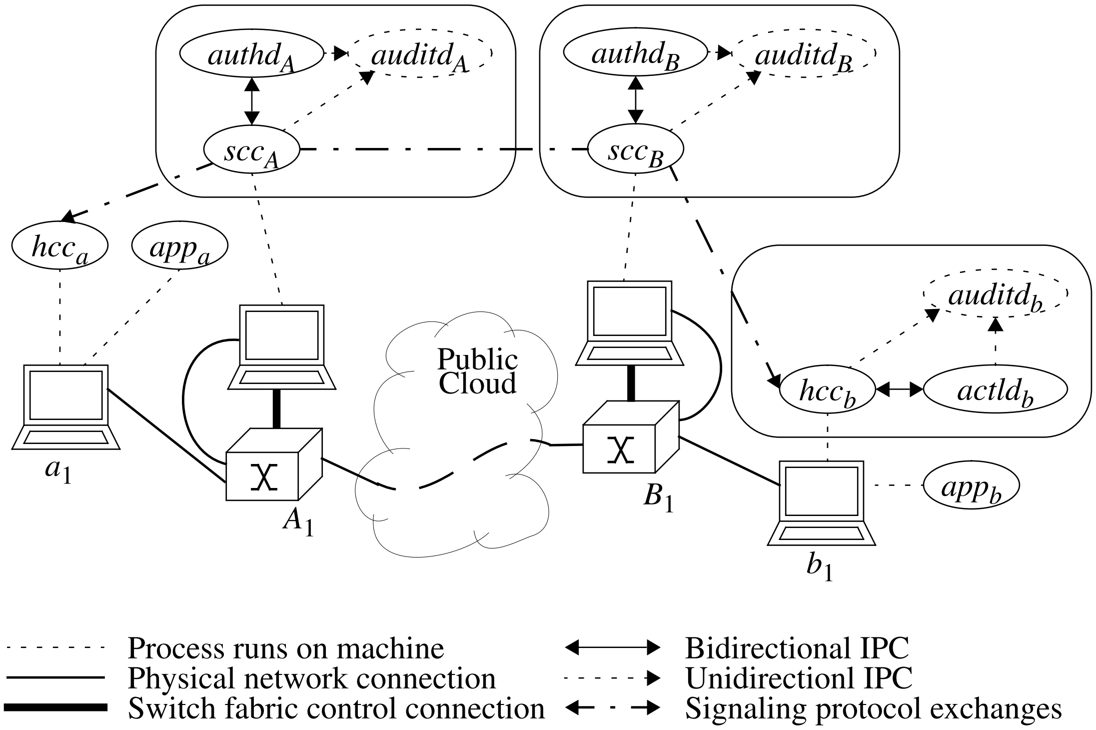

Figure 8.6 Example of a configuration in the implementation architecture: connection establishment from application app_a on host a_1 to application app_b on host b_1. Note that $authd_A$ performs the signing while $authd_B$ performs the verification of the information element.

- *scc*: *Switch call control* processes control switched virtual circuits on the switch side. They coordinate the setup and release of point-to-point and point-to-multipoint calls between Q.2931 interfaces. They handle requests for reservation, starting of data flow, and clearing of connections from the fabric control, obtain routes for incoming calls, respond to error messages from Q.2931 and the fabric control, and handle system initialization.

- *authd*: The *authentication daemon* provides an authentication service for connection endpoints (see section 5.2.1) and an integrity service for signaling messages (see section 5.2.2). It is used for both: signature generation and signature verification.

- *actld*: The *access control daemon* provides an access control decision service for signaled connection establishment (see section 5.2.3), i.e. call admission control.
- *auditd*: The *audit daemon* provides a system logging facility (e.g., syslogd) (see section 5.2.4).

The collection of *hcc* and *scc* processes at the various locations throughout the network comprises the connection management; the collection of *authd*, *actld*, and *auditd* make up the security controls as in figure 7.1. Process *authd* is sufficient to implement the security functions required for authenticated signaling. Processes *actld* and *auditd* are included in this design to illustrate the use of authenticated signaling for the second purpose (see section 7.2) and to follow the reference model for firewall technology (see chapter 5: *actld* is an example for an access control function ACF and *auditd* for an audit function AudF.)

8.4 Interaction of Processes

8.4.1 Interaction of Processes: End to End

Figure 8.6 serves as an illustration for how the processes interact. Process app_a on host a_1 attempts to send data to process app_b on host b_1. The two machines are connected via two switches over a public cloud, where switch B_1 represents the perimeter switch of the network domain to which destination host b_1 belongs. Application app_a communicates to hcc_a the request to establish an authenticated virtual circuit to app_b. This notification can take place explicitly when app_a informs hcc_a directly as in the original design of Q.port or semi-transparently for the application app_a as in our design. In both cases a connection is established through the execution of the distributed connection management algorithm (e.g., Q.2931) among the *hcc* and *scc* processes along the path of the connection. Because the establishment of an authenticated connection is desired, the *hcc* and *scc* processes interact with the authentication and access control servers.

Figure 8.6 can be seen as a snapshot of a system with two switches and several connected hosts, where switch B_1 serves as the perimeter switch. A more elaborate scenario would include a cascade of perimeter switches for a given domain (networks within networks), protected network domains on both the source and destination sides, and secure multicast communication (i.e., not only point-to-point). This example also depicts a distribution of services that offers scaling benefits as explained in section 5.3. The authentication function is provided by $authd_B$ on switch B_1, while the access control function is provided by $actld_b$ on the destination host b_1.

8.4.2 Interaction of Processes: Locally at Host

It is sufficient to augment a single protocol message, namely SETUP, to implement a unilateral authentication of the initiator of a point-to-point connection. As discussed in section 8.8.6 there are various reasons to protect other connection management protocol messages as well. For an investigation of authentication mechanisms it is sufficient to concentrate on one protocol message. The security services of mutual authentication and group authentication (for example for point-to-multipoint or multicast) merely require an application of the same mechanisms to a variety of protocol messages. We chose the SETUP message for the investigation.

Figure 8.7 (a) illustrates the interaction between applications and call control processes in more detail. Case (b) in figure 8.7 augments case (a) by interactions with library functions and security control processes (*authd* in figure 8.7). Data streams are denoted by heavy solid lines, call control interactions are denoted by thin solid lines, and connection management protocol exchanges are denoted by broken lines.

Case (a) in figure 8.7 depicts the arrangement of two user space processes (generic application *app* and host call control *hcc*) and the TCP/IP and ATM modules in the kernel of a host. If application *app* sends data to a remote destination to which no connection is established, the data cannot be forwarded past the point indicated by Ⓐ: first, a connection needs to be established. Conceptually, the connection

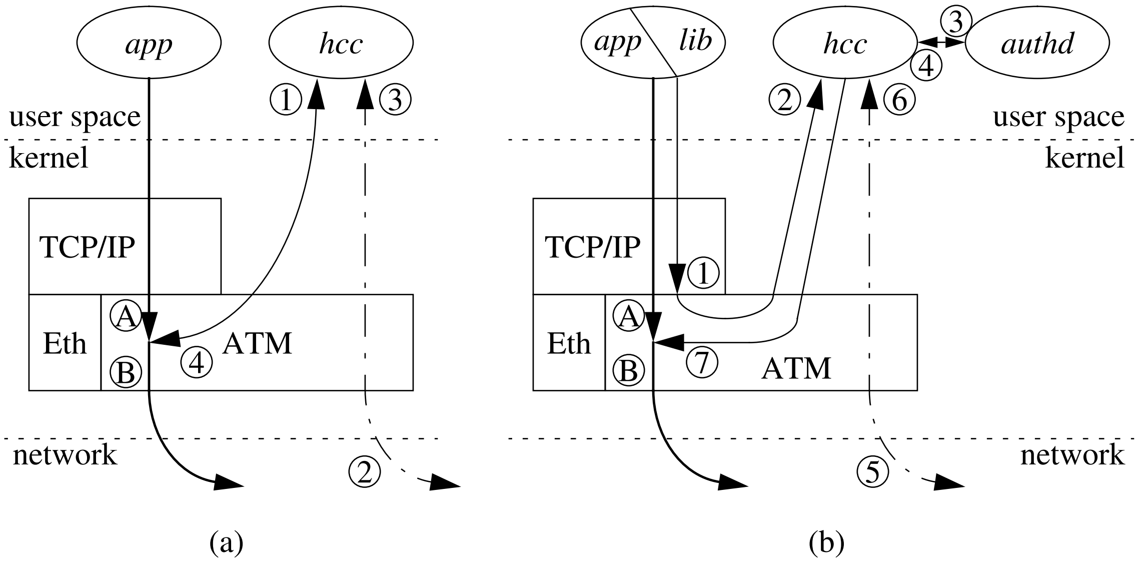

Figure 8.7 Interaction of components during connection establishment (a) without authentication and (b) with authentication.

establishment is part of the ATM layer, but in our implementation the call control processes are located in user space. This design requires the kernel to communicate the need for a connection establishment to the *hcc* process (①) that in turn executes the Q.2931 protocol to process the connection establishment request (② and ③). As soon as the ATM layer is informed by *hcc* that the connection is established (④), data can be transmitted at Ⓑ.

In our design, the connection establishment process happens transparently to the application. Connections are established on demand when data is present for a destination to which no connection exists. They can be torn down based on a variety of policies, such as "connection was not used recently."

Application *app* in figure 8.7 (b) sends data to a remote destination, but in this case it requires an authenticated connection. The authentication can happen transparently to the application only if there is no requirement for the application to provide cryptographic keys, such as if the kernel manages cryptographic keys on behalf of applications or if the granularity of authentication is on a per machine basis.

The application uses a library (*lib*) to facilitate the exchange of authentication information and to request and release authenticated connections. Linn ([Lin93]) offers a proposal for a *generic security service application program interface* (GSS-API) for the establishment of security associations. A library is not required if the kernel handles the authentication transparently.

Cooperation of applications is necessary if the authentication verification depends on authentication information that cannot be generated in the kernel. One-time passwords, such as those provided by smartcard authentication tokens (for example SecurID; [Rub96, §2.1]) or software solutions (for example S/Key; [Hal94]), serve as examples. Case (b) in figure 8.7 sketches our design where an application uses library calls to establish authenticated connections to destinations. If an application wants to send data over an authenticated pipe, it needs to request the authenticated call setup first (①). The ATM module relays the request to *hcc* (②). Process *hcc* contacts process *authd* for the creation of authentication signatures (③ and ④) before it engages in the distributed connection establishment protocol with its peer call control processes (⑤ and ⑥). Once the connection is established (⑦) data can be forwarded as in case (a) of figure 8.7. A limited amount of buffer space is available between Ⓐ and Ⓑ. In our implementation we used BSD UNIX-style mbufs for buffering IP frames. We did not experience buffer memory exhaustion during our experiments and testing. However, if much data is sent to the destination and there is considerable delay before a connection is established, we expect that memory can become a scarce resource.

The *inter process communication* (IPC) between the *hcc*/*scc* and the *authd*/*actld*/*auditd* processes can be implemented in a variety of ways. For our prototyping effort it was sufficient to use blocking IPC over UNIX sockets. Because we chose blocking IPC, no further Q.2931 protocol changes were necessary. The socket I/O (*input/output*) had to be added in two places in processes *hcc* and *scc*: at time of creation of outgoing SETUP messages, and at time of receipts of incoming SETUP messages. Module configurations as described in 8.6 determine if signature creation or signature verification actions need to be performed.

This design has the advantage that the *authd* process can perform its cryptographic calculations directly on the signaling IEs without having to recreate them, and it avoided a problem that occurs if the *authd*-type functionality is added too low in the Q.2931 stack: that receivers can interpret normal message retransmission as replay attacks ([PT97]). Our architecture can support a coexistence of several authentication IEs within one signaling message (e.g., for various security levels) as well as nested authentication.

8.5 Authentication Information Element

This section describes our design of a Q.2931 information element to convey authentication (and access control) information between principals that participate in the connection establishment process. We define the *authentication, access control, and audit information element* (`aaaIe`) according to section 5.4.5.1 and figure 5-23 in [ATM94].

Information elements are exchanged as an octet stream in the same format they are handed to the *signaling ATM adaptation layer* (SAAL) for transmission to peer signaling entities. Their format is defined in a standard's document ([ATM94]).

The information element represents one particular example of how one can encode the data. There are other possibilities, such as the ones described in [TPB+96, section 8.1.1] and [LR97].

Table 8.1 lists the coding of the header of the `aaaIe` according to [ATM94, figure 5-23] as well as the space allocated for the body. In our prototype the body consists of an octet array of up to 508 octets. This size was chosen to create an octet array large enough to allow for experimentation with its contents. This prototyping technique allowed us to transmit a block of octets transparently to the signaling protocol. At the places where the contents of the body need to be interpreted, the coding structure (such as the example in figure 8.2) is overlayed. Once the signaling protocol

byte	coding	meaning
00	fe	information element identifier (0xfe is an arbitrary choice)
01	80	bit 8 ext=1
		bit 7-6 = 00 - coding standard: ITU-T
		bit 5 flag = 0 - in agreement UNI 3.1
		bit 4 = 0
		bit 3-1 IE action indicator = 000 - in agreement UNI 3.1
02-03	01 fc	0x01fc = 508_d size of IE. Total 512 bytes.
04-1ff	xx xx	508 bytes available for the authentication value

Table 8.1 AAA information element — header and octet array for body

implementation was modified to transmit the information element, we could refine the contents of the information element gradually without having to interfere with the running signaling protocol processes, minimizing the impact of our experimentation on other uses of the network.

Section 8.8.5 presents four categories of information that are part of an authentication information element in our architectural design. The fields in the body of the `aaaIe` (table 8.2) are part of these categories as follows:

- *Meta Information*: `version`, `protocol`, `hash alg.`, `encryption alg.`
- *Algorithm Specific Information*: `nonce_no`, `nonce_time`, and `signature`
- *Identification Information*: the remaining fields in table 8.2 were used for the identification of the participating principals. Fields labeled `source...` or `calling_...` identify the sender and are used for source endpoint authentication.

  Our choice of identification information displayed in table 8.2 contains data fields which require "layer violations" for values to be assigned to them (e.g.,

name	len	type	description
`version`	1	`u_char`	Version of this Ie layout
`protocol`	1	`u_char`	protocol identifier
`nonce_no`	4	`long`	Nonce number
`nonce_time`	8	`long[2]`	Nonce Timestamp (NTP format)
`hash alg.`	1	`u_char`	hash algorithm used
`encryption alg.`	1	`u_char`	encryption (signature) algorithm used
`destination NSID`	1	`u_char`	destination name space identifier
`source NSID`	1	`u_char`	source name space identifier
`destination ID`	4	`u_int`	ID of receiver
`source ID`	4	`u_int`	ID of sender
`destination GID`	4	`u_int`	GID of receiver
`source GID`	4	`u_int`	GID of sender
`destination socket`	16	`struct sockaddr`	Socket address of receiver
`source socket`	16	`struct sockaddr`	Socket address of sender
`called_atm_len`	1	`short`	ATM address of receiver
`called_atm_addr`	20	`u_char*`	
`called_sub_len`	1	`short`	ATM subaddress of receiver
`called_sub_addr`	20	`u_char*`	
`calling_atm_len`	1	`short`	ATM address of sender
`calling_atm_addr`	20	`u_char*`	
`calling_sub_len`	1	`short`	ATM subaddress of sender
`calling_sub_addr`	20	`u_char*`	
`signature`	200	`char[200]`	Cryptographic signature of above data
Total length	$\Sigma = 350$		

Table 8.2 AAA information element — example coding for body portion of `aaaIe`

`destination socket`). These fields were chosen for our prototype and test applications (see section 8.2.2 for a description). In a more general approach one could use the BHLI IE as a means for exchanging high layer information and as an encoding to be included into the signature.

- *Message (for integrity protection)*: the same fields that were used as identification information were also protected against modification. Fields labeled `destination...` or `called_...` identify the intended destination and can be used in an access control decision.

  We did not include other IEs of the Q.2931 protocol into the calculation of the cryptographic signature. If one were to do that, IEs which need to be covered by the signature would either have to be included into the authentication IE, or there would have to be an encoding or a standard (specifying their selection and order) to determine those IEs.

8.5.1 Information Encoding

All address and message fields have an associated length field and contents are padded with 0x00 to maximum length. Eventually, the same coding as for information elements is likely to be used for data members `senderid`, `receiverid`, ..., `calling_sub_addr`, but for our prototype it was sufficient to replicate the information.

Timestamps t_i are represented in the format of the *network time protocol* (NTP). To quote the standard ([Mil92, section 3.1]):

> "NTP timestamps are represented as a 64-bit unsigned fixed-point number, in seconds relative to 0h on 1 January 1900. The integer part is in the first 32 bits and the fraction part in the last 32 bits."

For portability reasons, we chose the exchange of information encoded in the standard presentation format over its exchange in a customized encoding of internal data structures native to the particular signaling implementation in use (e.g., Q.port).

8.6 Module Configuration

Once the mechanism of authenticated signaling is available, security enforcement becomes a configuration issue. Figure 8.8 repeats the configuration of the security

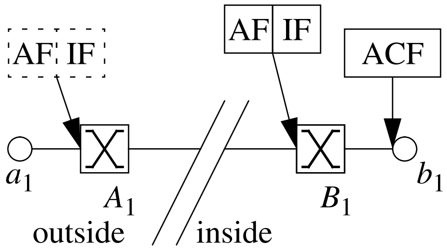

Figure 8.8 Distribution of functional blocks

functions of figure 8.6 at a more abstract level (in the notation used in section 5.3). In this example the authentication (integrity) function at switch A_1 creates the authentication signature (digital signature, respectively) on behalf of principal a_1. The authentication and integrity functions at switch B_1 verify authentication signature and digital signature. The *access control function* (ACF) is performed at the end host b_1.

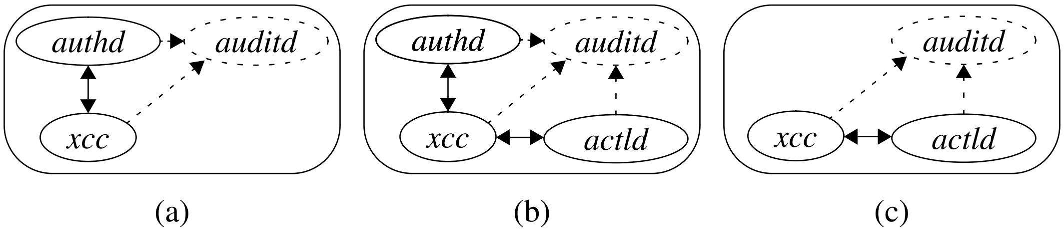

Figure 8.9 Module configuration for AF (and IF), ACF, and AudF functionality at process xcc, where $x \in \{h, s\}$. AF and IF are implemented by *authd*, ACF by *actld*, and AudF by *auditd*. (a) xcc invokes an authentication action (e.g., generation of verification of a digital signature). (b) xcc invokes an authentication verification action and requests an access control decision; (c) xcc requests an access control decision without preceding authentication.

Figures 8.6 and 8.8 illustrate that the software modules for authentication and access control can be configured in various ways. Figure 8.9 displays the three possible configurations. Process xcc in figure 8.9 can be replaced either by hcc or by scc.

Case (a) displays a configuration in which *xcc* communicates only with *authd*. This configuration could be on the sender or receiver side of the communication. Consider a unilateral authentication in figure 8.6 based on digital signatures. The location of these processes determines if *authd* needs to perform the generation or the verification of digital authentication signatures. Server instance $authd_A$ needs to generate digital signatures because it serves the initiator of the connection. The server instances $authd_B$ and $authd_b$ verify digital signatures because they serve the target of the connection. In a mutual authentication scenario *authd* processes can perform both actions: generation and verification.

Case (b) displays a configuration in which both authentication and access control are performed. In case (c) only the access control daemon is accessed. On the sending side the access control check can be the decision if the local initiator is authorized to make a connection which leaves the local network policy domain. On the receiving side it can be the decision if remote traffic is allowed to enter the local network policy domain.

	I: initiator side	T: target side
(a)	Generation of authentication and integrity information	Verification of authentication and integrity information
(b)	Combined generation of authentication and integrity information and access control decision for connection leaving network policy domain	Combined verification of authentication and integrity information and access control decision for connection entering network policy domain
(c)	Access control decision for connection leaving network policy domain	Access control decision for connection entering network policy domain.

Table 8.3 Summary of configuration options

Table 8.3 summarizes the various configuration options. Independent of the configuration, processes *xcc*, *authd*, and *actld* all have write access to the audit module to log information according to the security policy in force.

8.7 Authentication Protocols

As we explain subsequently, augmenting Q.2931 with authentication and integrity assurance can be achieved through the choice of an authentication protocol and the transmission of one additional information element. Burrows, Abadi, and Needham ([BAN89] and [AN94]) document a number of authentication protocols in which flaws have been discovered and exposed after years of their use. Some of these flaws were so serious that the protocols could no longer be used for their purpose. [BAN89] and [AN94] conclude that choosing an existing authentication protocol that is published, that has been scrutinized for many years, and that has not been broken is preferable over the attempt to design a new authentication protocol. By mid 1997 it is considered "good practice" in system design to make the choice of cryptographic protocols a parameter of the system using them (e.g., [AMP96, §1.3]).

The following sections describe and discuss two examples of authentication protocols which are suitable for integration with the signaling protocol Q.2931 (appendix A for a description of the notation of cryptographic protocols). The authentication protocol fulfills the authentication function AF (of signaling endpoints) and the integrity function IF (of signaling messages) as described by the reference model in section 5.1. A variety of other authentication protocols are possible. They can consist of a combination of signature schemes, identification schemes, and authentication codes ([Sti95, chapters 6,9,10]). Two examples are used subsequently to explore design issues of authenticated signaling.

8.7.1 Two Examples of Authentication Protocols

The first protocol is based on digital signatures, where a hash of the authentication protocol message is public-key encrypted with the private key of the sender and sent together with the authentication message (see figure 8.10). The second protocol is based on keyed hashing (see figure 8.11). Both protocols authenticate principal a and ensure the authenticity and integrity of certain portions of the authentication message, denoted by m. Table 8.4 lists the time stamps used in the example authentication protocols. The verification steps for both protocols are identical and are summarized in table 8.5).

time t_1	: a begins creating the authentication protocol message
time t_2	: b has received the authentication protocol message
time t_Δ	: time window in which only different sequence numbers are accepted

Table 8.4 Time stamps used in the example authentication protocols

8.7.2 First Protocol: Based on Signed Hashing

The first protocol is based on signed hashing, as used in IETF standards (for example [BCCH94, §3.3.2] and an explanation in [KPS95, §9.2.4]).

8.7.2.1 Protocol

In this protocol (figure 8.10), principal a, for example the initiator (i.e., claimant, sender) of a connection, creates a message digest in step (1) that contains the identities of a and b, timestamp t_1, nonce n_a, and possibly other information, denoted here by m. In step (2) a signs the hash value with his private key K_a^{-1}. Step (3) represents the transmission of the signature and the original data. The transmission is done by encoding this information for inclusion into the information element and transferring it with the Q.2931 SETUP message. Step (4) illustrates that the receiver b needs to look up the public key K_a of a before he can continue processing the authentication verification. Steps (5) and (6) could be exchanged because their order does not matter

step	src		dst		protocol message
(1)	^{t_1}a			:	$h_a := \mathcal{H}(m, t_1, n_a, a, b)$
(2)	a			:	$s := \{h_a\}_{K_a^{-1}}$
(3)	a	$\rightarrow$	$b\ ^{t_2}$	:	$(m, t_1, n_a, a, b, s) \rightarrow (m^*, t_1^*, n_a^*, a^*, b^*, s^*)$
(4)			b	:	lookup K_a
(5)			b	:	$h_b := \{s^*\}_{K_a}$
(6)			b	:	$h_a^* := \mathcal{H}(m^*, t_1^*, n_a^*, a^*, b^*)$

Figure 8.10 Protocol based on signed hashing

in this example. In either case, b needs to recreate a hash value of the received information (h_a^*) and decrypt the received signature s^* to prepare for their validation (see section 8.7.4). The asterix as a superscript (*) denotes that values transmitted over the network may have been modified by an active wiretapper.

After successful execution of the authentication protocol, principal a has established his authenticity with principal b (the verifier/receiver) and assured the integrity of data message m. The authentication message consists, for example, of a data message, a timestamp, a sequence number, and identifiers for the participants of this protocol, a and b. The data message m can be empty if only the authenticity of a is desired. If m is not empty, this protocol establishes its integrity upon successful execution. For example, the data message can contain the first n octets of the first IP packet for this connection and a combination of information elements. The exact contents, coding, and layout of an example authentication message are presented in section 8.5.

8.7.2.2 Assumptions

This protocol assumes that clocks between claimant and verifier are synchronized, that the private key of the claimant is not compromised, and that a secure public key infrastructure exists, such as [CCI88]. K_a, the public key of principal a is a public value. It may be cached for improved performance for future connections with the same initiator. Caching issues are outside the scope of this dissertation.

8.7.3 Second Protocol: Based on Keyed Hashing

The second protocol is based on Diffie-Hellman key exchange (see [DH76]). The key generation is similar to the proposal for the *Simple Key-Management for Internet Protocols* (SKIP) ([AMP96]).

8.7.3.1 Assumptions

It is assumed that the private values i and j of the sender and receiver and the shared master key, once it is calculated, are not compromised, and that the public values $g^i \bmod p$ and $g^j \bmod p$ are authenticated public values. In addition to these public values, the common secret master key K_{ij} should be cached for future speedup.

8.7.3.2 Protocol

In steps (1) and (2) principal a generates the shared master key K_{ij} in Diffie-Hellman style. The session key K_{ijn_a} is generated in step (3) the same way as proposed in SKIP (see [AMP96, §1.2] for a discussion). K'_{ij} denotes the low order 256 bits of K_{ij}. Step (4) illustrates the formula for authentication key generation (see [AMP96, §1.9] for a discussion). The message digest h_a is generated in step (5) differently than in the first protocol (steps (1) and (2) in figure 8.10). It does not need to be encrypted. Rather the secret session authentication key K_{Auth} is made part of the input to the hash function. Short of breaking the hash algorithm, an active wiretapper cannot generate a valid hash h_a^* for a forged message without the secretly shared authentication key.

On the receiving side, principal b needs to generate the keys as did a before: K_{ij}, $K_{ijn_a}^*$, and K_{Auth}^* (Steps (8)-(10)). Principal b recalculates the signature h_b in step (11) for later verification (see section 8.7.4).

step	src		dst		protocol message
(1)	a			:	lookup $g^j \bmod p$
(2)	a			:	$K_{ij} := g^{ji} \bmod p$
(3)	a			:	$K_{ijn_a} := \mathcal{H}(K'_{ij}\|n_a\|0x01) \mid \mathcal{H}(K'_{ij}\|n_a\|0x00)$
(4)	a			:	$K_{Auth} := \mathcal{H}(K_{ijn_a}\|\text{AuthAlg}\|0x03) \mid \mathcal{H}(K_{ijn_a}\|\text{AuthAlg}\|0x01)$
(5)	^{t_1}a			:	$h_a := \mathcal{H}(K_{Auth}, \text{keyfill}, m, t_1, n_a, a, b, K_{Auth})$
(6)	a	$\rightarrow$	$b\ ^{t_2}$	:	$(m, t_1, n_a, a, b, h_a) \rightarrow (m^*, t_1^*, n_a^*, a^*, b^*, h_a^*)$
(7)			b	:	lookup $g^i \bmod p$
(8)			b	:	$K_{ij} := g^{ij} \bmod p$
(9)			b	:	$K^*_{ijn_a} := \mathcal{H}(K'_{ij}\|n^*_a\|0x01) \mid \mathcal{H}(K'_{ij}\|n^*_a\|0x00)$
(10)			b	:	$K^*_{Auth} := \mathcal{H}(K^*_{ijn_a}\|\text{AuthAlg}\|0x03) \mid \mathcal{H}(K^*_{ijn_a}\|\text{AuthAlg}\|0x01)$
(11)			b	:	$h_b := \mathcal{H}(K^*_{Auth}, \text{keyfill}, m^*, t_1^*, n_a^*, a^*, b^*, K^*_{Auth})$

Figure 8.11 Protocol based on keyed hashing

8.7.4 Authentication Verification

After the last step of either protocol is completed, principal b performs a number of tests to verify the authenticity of principal a, and the authenticity and integrity of the signed data message m. The authenticity of a and the integrity of m are not established if a single test fails.

Principal b compares the two message digests h_a^* and h_b. If they are not equal the authenticity of a cannot be established. This comparison (1) can detect attacks, such as active wiretapping. For the second protocol, h_a^* and h_b are signatures, not merely hash values.

The second test (2) allows principal b to release connection attempts which were originally intended for a separate destination (see [Atk95a, section 6]).

Test (3) protects principal b against replay attacks. Connection attempts are only accepted within a time window t_Δ.

	evaluates to *true*	result
(1)	$(h_a^* \neq h_b)$	signature mismatch
(2)	(identity of receiving node $\neq b^*$)	destination mismatch
(3)	$(t_1 \notin (t_2 - t_\Delta, t_2])$	timing violation
(4)	(n_a has been seen by b in t_Δ)	sequence number mismatch

Table 8.5 Verification procedure for authentication protocols

Principal b caches the tuples (x, n_x) for each sequence number n_x which was seen in the last t_Δ time. If b receives an authentication protocol message from a, b ensures that n_a is not identical to any of the cached sequence number values for a: test (4). This test allows for the concurrent execution of several instances of the authentication protocol between two principals within a given time interval. Principal b does not have to keep a history for each possible peer of the largest sequence number which has been used in the past to prevent replay attacks. Principal b only needs to keep sequence numbers in its cache that are valid within the most current time window of length t_Δ. (See [Ran93a, Annex F.1] for a discussion of unique number mechanisms with on-line authentication certificates.) Under the assumption that principal a will never reuse the same time stamp t_1, the time stamp itself can be used as a nonce.

8.7.5 Other Examples

Three more examples of proposals for authentication protocols can be found in [TPB$^+$96] and [Tar97a]. Tarman et al. ([TPB$^+$96]) describe a protocol based on the *digital signature standard* (DSS) ([NIS94]). The ATM Forum proposes in [Tar97a] two protocols based on the ISO/IEC standards 9594-8 and 11770-2.

8.8 Exploration of Design

8.8.1 Public Key vs. Private Key Cryptography

The first protocol is based on public key cryptography, the second on symmetric key cryptography. In the first protocol the verification step can be performed at various places in the course of the connection setup, e.g., at a signaling process participating in the connection setup. This is because only public keys and no shared secrets are required for verification. This feature allows security policy enforcement anywhere along the path of the connection, in particular at the network perimeter and is one of the benefits that authenticated signaling offers over other methods. To achieve the same characteristic with the second protocol requires the destination of the connection attempt to share the master key with intermediate policy enforcing nodes.

The second protocol has a performance advantage over the first protocol: in the second protocol, the computationally expensive exponentiation operation has to be performed only once per peer (protocol steps (2) and (8)), not once per connection attempt because the master key K_{ij}, once calculated, can be cached for future use. This approach allows for smaller overhead in the connection establishment phase if more than one connection originates from the same initiator. The use of hardware devices for the computation of cryptographic functions can yield a speedup over software solutions (for example, [Ebe92]). Peyravian and Tarman ([PT97]) discusses further aspects of the trade-offs between public key and secret key authentication in ATM signaling protocols.

8.8.2 Clock Synchronization

Authentication protocols can be vulnerable to a certain class of timing attacks if they are based on synchronized clocks and the assumption of clock synchronization is temporarily violated (see [Yah94]). Both protocols assume the principals' clocks to be synchronized. We consider this assumption to be reasonable because by mid

1997 precise clocks have been considered standard components in computing systems for several years. Secure network time protocols exist and are in widespread use. SNTP ([Mil96]), for example, is capable of delivering time accurate to the order of microseconds. Even larger clock drifts are acceptable as long as t_Δ is chosen accordingly. One should choose t_Δ to be slightly larger than twice the maximum round trip time between possible participants, plus the processing time required for the execution of the applicable authentication protocol (for a discussion on the choice of t_Δ see [KPS95] or [Sti95]).

Resynchronization of clocks does not have to depend on authenticated connections because the time signal itself can be authenticated by known mechanisms (see [Mil92, Appendix C]). Therefore, we do not have to cope with a bootstrap problem to recover the correct time in case clocks do lose synchronization, through, for example, system failures.

8.8.3 Number of Protocol Messages

Note that both protocols require only one protocol message to be sent from the initiator (a) to its peer (b) for unilateral authentication, which implies that two protocol messages (one in each direction) are sufficient for mutual authentication of unicast traffic. This feature enables the use of two-way handshake connection management protocols for transport of the authentication protocol messages for mutual authentication (in contrast to a requirement for a three-way handshake for mutual authentication based on a challenge response mechanism; [Yah94] and [KPS95, §2.4.4]).

The choice of which authentication protocol to superimpose on the connection establishment process depends on the number of messages required for the connection establishment. Modifying the protocol message flow by adding, deleting, or rearranging protocol messages is a major change to a protocol. Such a modification goes far beyond our proposal of using the connection management protocol for transport of security protocol messages. It is outside the scope of this dissertation. Stallings presents an overview of authentication protocols for network security in [Sta95].

8.8.4 Timing of Authentication Verification

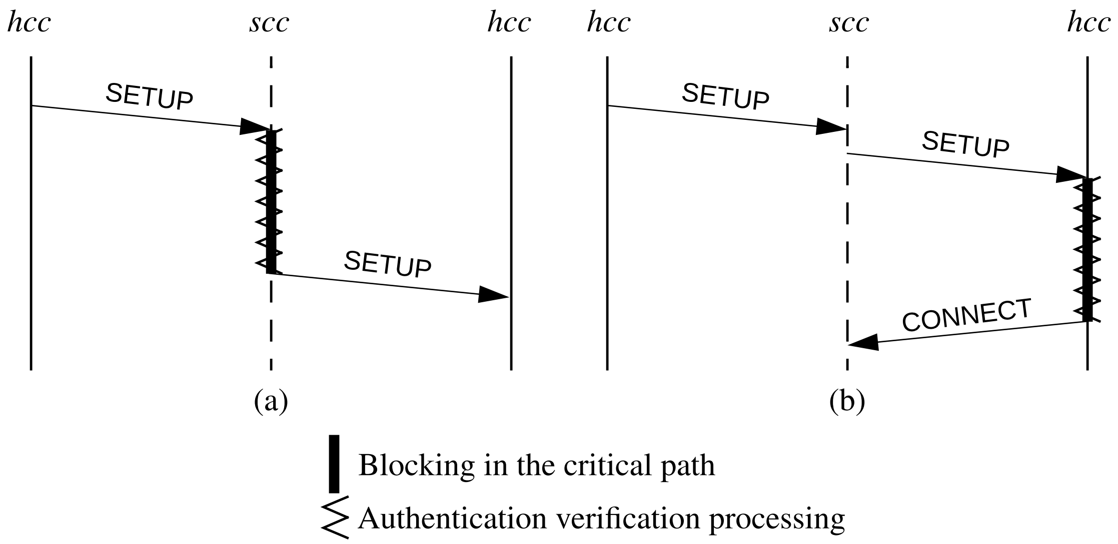

Figure 8.12 Illustration of blocking during serial verification of the authentication during ATM connection establishment. (a) The authentication verification is performed by an intermediate *scc* before the SETUP message can proceed towards the destination. (b) The authentication verification is performed by the destination *hcc* before the CONNECT message is sent.

There are various opportunities for the verification of an authentication protocol message to take place. This section discusses the placement of authentication verification and its implications for possible blocking in the critical timing path of connection management protocols.

One possibility is to verify the authentication message before a SETUP is forwarded to the successive *scc* or *hcc* process (see figure 8.12 (a)), or, in case *hcc* performs the authentication verification, before a CONNECT is sent by the receiver (see figure 8.12 (b)). Both choices grant priority to security concerns because protocol processing is delayed until the outcome of the verification is known. They have, however, the disadvantage that the computationally expensive verification step is in the critical timing path of every secure connection establishment. Alternatives are available:

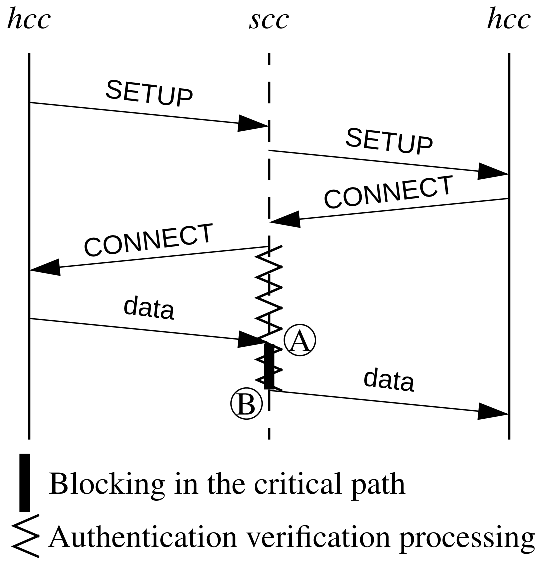

Figure 8.13 Illustration of verification of the authentication after an ATM connection is established, but before any data is permitted through the connection. An intermediate *scc* performs the verification after the CONNECT message is sent. Any received data needs to be buffered (at Ⓐ) until the verification is completed (at Ⓑ).

- After receipt of the SETUP message the destination proceeds with the connection setup as usual. The verification will be processed only after the CONNECT is sent (see figure 8.13). Data received by the destination through the established channel is queued and not forwarded to its destination until the verification is completed. If the verification fails, the connection is released and the queued data cleared. If the verification succeeds, data that has accumulated in the waiting queues is forwarded to its higher layer destination. This approach has drawbacks: queue management requires additional resources and processing. Furthermore, this approach is not feasible if intermediate switching processes are supposed to authenticate the source of a connection before a SETUP message is forwarded into the guarded network.

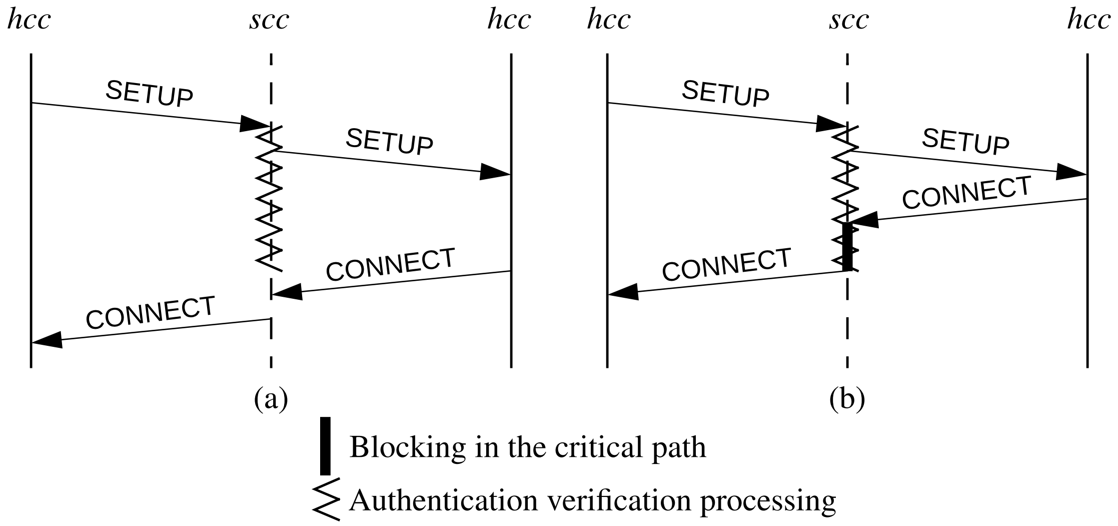

Figure 8.14 Illustration of verification of the authentication concurrently during connection establishment. An intermediate *scc* starts the verification after the SETUP message is received. (a) If the result of the verification is present by the time the CONNECT message is received by *scc* no additional delay is incurred. (b) Otherwise *scc* blocks until the result of the verification is present before it sends successive CONNECT.

- After receipt of a SETUP message, the intermediate verifying node initiates the authentication verification and processes it concurrently to the connection setup. When the CONNECT message from the destination for this connection reaches the verifying node, the authentication verification process has either been completed (see figure 8.14 (a)), or not (see figure 8.14 (b)). In the former case, the signaling protocol experiences no additional delay. In the latter case, the signaling cannot proceed until the verification process has finished. From then on, above discussion about the success or failure of the verification applies. The advantage is a potential decrease in connection establishment latency through concurrent processing of the authentication verification and the connection establishment protocol compared to the previously discussed approaches.

8.8.5 Authentication Information

The state and availability of a network can be affected by control plane messages. The integrity of all possible signaling messages, or a subset thereof, need to be protected, depending on the level of assurance that is desired. For unilateral authentication of a connection initiator it is sufficient to concentrate on SETUP messages. If mutual authentication is desired, CONNECT messages need to be protected; if protection against denial of service through unauthorized connection clearing is desired, RELEASE and RELEASE COMPLETE messages need to be protected; if protection of point-to-multipoint connections is desired, the according protocol messages (ADD PARTY, DROP PARTY ACKNOWLEDGE, etc.) need to be protected.

To provide maximum protection, a signaling protocol needs to provide the capability of assuring the integrity and authenticity of all its signaling messages. Note that our focus is on connection signaling: other services, such as routing updates, or name resolutions are outside the scope of this analysis and have their own security requirements.

It is not sufficient to sign signaling messages transparently as a whole: some parts of signaling messages cannot be included in an end-to-end protection of messages because they may be modified by intermediate signaling processes which are not capable of recalculating valid signatures. If such parts were included in the calculation of the message digest, the destination could not verify the digest because it could not recreate the message. Examples include information elements which are only present at certain interfaces, such as the "transit network selection," and message header fields which may change, such as "message length." The purpose of the "transit network selection" information element is to identify one requested transit network. It can only appear at the UNI in the user to network direction. Once the "transit network selection" information element is removed from a signaling message by some *scc* process, the "message length" needs to be decreased by its length.

We mentioned before that the security protocol message can be transmitted in an information element. Based on our design, the following categories of information need to be present in such an authentication information element:

- *Meta Information.* For example, authentication or hash algorithm identifiers. Meta information enables protocols to be independent of the choice of cryptographic algorithms as motivated in section 8.7.
- *Algorithm Specific Information.* For example, nonces, timestamps, digital signatures.

In addition there are two more categories of information necessary that do not necessarily need to be transmitted in the authentication IE.

- *Identification Information.* For example, source and destination ATM addresses, IP addresses, port numbers, *user/group identifiers* (UID/GID).
- *Message (for integrity protection).* For example a set of information elements present in a connection management protocol message.

Identification information is required to identify the principal that is being authenticated (and bound to the cryptographic keys used for his authentication). The message whose integrity is to be protected is also necessary. Connection management protocol messages, such as Q.2931, contain both these two categories of information, so they do not need to be included in the authentication information element a second time. There are cases, however, where they are only partially present: for example, cryptographic keys can be bound to higher layer principals than the network interface address that serves as the connection endpoint on their behalf. In that case a verifier requires more detailed identification information than presented by the connection management protocol. Abadi and Needham ([AN94]) presents a principal of cryptographic protocol design that makes this point:

> "If the identity of a principal is essential to the meaning of a message, it is prudent to mention the principal's name explicitly in the message."

The principle requests that the identification information should appear in the authentication information element if it is not explicit in the rest of the connection management protocol message.

8.8.6 Protection Against Threats

The authentication of signaling messages allows the ATM control plane to verify the source and contents of a signaling message before acting on it. It therefore protects the network infrastructure from a number of threats and possible attacks ([Den82, §1.2]), such as masquerading, active wiretapping, or denial of service. An example of a denial of service attack is the sending of forged RELEASE messages for established connections. In addition to this protection, authentication can be the basis for services, such as non-repudiation and billing.

Threats of certain other types of denial of service remain present, even in the presence of strong authentication. For example, nothing prevents a culprit from starting the connection establishment of a large number of connections without ever completing them. In some network protocols, such as TCP or unauthenticated Q.2931, a culprit can even spoof the source addresses of authentication establishment messages and minimize the chance of discovery ([SKK+97, CER96]). Typically, protocols rely on timeouts to recover from attacks as these. However, the timeouts were introduced to recover from occasional network reliability problems and not to protect against dedicated denial of service attacks ([SKK+97]). Available mechanisms are not sufficient to protect against this class of attacks. In the presence of authenticated signaling, another resource is under attack: the computation resource calculating the verification of the "authenticated" information elements. If authentication verification becomes unavailable through a denial of service attack, legitimate connections cannot succeed because their validity can no longer be verified.

8.8.7 Further Security Considerations

There are concerns by telecommunication service providers that additional information elements potentially provide end users with free bandwidth: the ability of transmitting data while avoiding billing charges. For example, a sender could encode data in the authentication information element and attempt the establishment to a destination with whom he colludes. The destination would retrieve the data received in the authentication information element, but not establish the connection, and thus avoid billing charges. Such fraud schemes are of concern to telecommunication providers ([Lyl94, Lyl95]).

Research on covert channels shows that there is always an opportunity to transmit data as long as a transport service is available, even in the presence of active countermeasures ([OOS$^+$97]). In the general case it is impossible for intermediate network nodes to determine if a connection attempt failed because the called party was not able to take the call or because of intermittent failures or security enforcement: there is not enough information available to intermediate nodes to make this distinction. Even if customers can be convinced that charging in the latter case is acceptable, charging in the former case is a considerable deviation from practice in many years preceding mid 1997 and financially disadvantageous for customers. We learned that the ability to authenticate all incoming signaling messages can be so important to some customers, such as national laboratories that require a high assurance of secure communications, that billing surcharges become acceptable to them ([Tar97b]).

8.9 Experimental Configurations and Demonstrations

We used a variety of test configurations. Minimal configurations included one switch and two hosts. The largest configuration consisted of three switches and up to six workstations. For our experiments we used two applications.

The first application was a client-server program which could be used to transfer strings. This application was sufficient for a proof of concept. The second application was a popular network video application, called *nv*. To use our prototype, it was necessary to add less than ten lines of code to the source code of nv, to compile it, and to link it.

For a demonstration of our prototype at PARC we equipped several workstations with cameras and used nv to transfer bi-directional video streams. The required connections were established and released on demand over the ATM network. Connection establishment was authenticated, rudimentary access control was performed, and system actions were audited to syslog according to configuration.

8.10 Performance Remarks

We have not included performance measurements detailing the efficiency of our implementation. The performance of cryptographic operations depends on a variety of factors, such as the choice of algorithms, key length, message size, implementation in software or hardware, exploitation of parallelism, and others. The particular combination of choices we made in this multidimensional space were guided by the desire to achieve proof of concept, functional correctness, and robustness.

Our implementation choices (for example the choice of programming languages and the decision to use a software implementation for computationally expensive cryptographic operations) are unlikely to match the choices designers would make for a production system. Although we did not optimize for raw speed, we rarely experienced situations during our experiments where authentication protocol overheads caused timeouts of Q.2931 timers, which in turn forces a retransmission of protocol messages.

8.11 Chapter Summary

This chapter described the realization of the concept of authenticated signaling in an ATM environment. The realization is a distributed system with interacting processes for connection management, as well as authentication, access control, and audit processes for a variety of experimental authentication protocols. These services are structured in the design according to the reference model in chapter 5. The resulting mechanisms can be used as a firewall security mechanism to address the problems as described in section 7.1.

The choice of a layout for an authentication information element is central to our implementation design. This chapter described the features of the authentication information element we designed for use in the prototype. The system was implemented, demonstrated, and explored as part of the Q.2931 B-ISDN signaling protocol in a heterogeneous hardware and software environment.

9. SUMMARY, CONCLUSIONS AND FUTURE DIRECTIONS

This dissertation presents a framework for firewall technology within which firewall systems can be designed and validated. It takes firewall technology out of a state where progress is made primarily through the reaction to the latest network-based attack and advances it into a state of careful architectural design as a proactive approach to network access control. This dissertation illustrates the potential of firewalls as network security devices that provide high assurance by design and implementation rather than by ad hoc engineering.

As an overall framework we present the life cycle of a firewall system in a fashion similar to a waterfall model. Its phases are supported by several methods. The following two methods are original contributions of this dissertation to the field of firewall technology:

The first method presented is a reference model for firewall technology (chapter 5). It was developed using historical research (through the examination previous work; chapter 3) and an examination of the question: what functionality is desirable for network access control, and what are its prerequisites? The reference model gives designers an understanding of which functions are desirable in a firewall system, how they can be composed to build a firewall system, how they interact, and how they need to be enforced. The model is therefore descriptive and prescriptive. The model does not specify how the functions are to be implemented. The essential components of the reference model are authentication, integrity, access control, audit, and their enforcement. The components can be deployed in a distributed fashion to achieve scaling. A further advantage of their distribution is their improved overall availability and protection through redundantly deployed mechanisms.

The second method presented is a design tool for firewall components and firewall systems alike (chapter 6). It is based on a formalism that uses *Hierarchical Colored Petri Nets* (HCPN, short CPN) to describe the functionality of mechanisms used by firewall technology. HCPNs provide us with a theoretical framework and means of description, composition, simulation, and analysis of firewall systems.

Chapters 7 and 8 describe the concept of authenticated signaling and the design, implementation, and exploration of its realization in ATM networks. Authenticated signaling can provide firewall security services and contribute as a building block to the construction of firewall systems. It is applicable to connection-oriented networking technologies. It is based on the idea that connection management protocols can provide the transport service required for security protocol messages to be exchanged. Authenticated signaling can be used to secure the connection management itself (signaling security), and to provide security services for a variety of other components in a network (security signaling), i.e., to all entities that can interface with the signaling protocol. Its services can be accessed at intermediate as well as connection endpoints. The reference model from chapter 5 guided the design and structure of the realization of authenticated signaling in ATM networks.

9.1 Experiences

The reference model illustrates an insight that we then confirmed through experimentation with our prototype of authenticated signaling: network access control services do not need to be provided at the network perimeter (the classical bottleneck of firewall systems) as long as they are enforced on each possible path between the network perimeter and the target of the communication. We could observe the scaling advantages of the latter approach in our experimental setup. This feature allows the technology to remain applicable in high performance networking technologies. One advantage of enforcing security at the network perimeter is that the network as a resource is protected against denial of service. Attacks that are blocked at the firewall might seal off the protected network from the rest of the world, but communications within the network can continue to operate normally. The more security services are moved closer to the targets of the communication traffic and away from the network perimeter the less the network is protected as a resource. The extreme point is the scenario of what is popularly called "perfect host security" at the cost of no protection of the resource network.

The reference model demands that certain security critical functions are performed before communication traffic reaches its target service. This requirement includes enforcement by target services themselves ([Ran97]).

Likely future trends in computer networking are addressed by the model. It can be applied to networking technologies, such as those required by wireless computing (the difficulty here is the definition of the network perimeter) and high performance networking as described above. We expect the reference model to have an educational and a guiding influence on the design of future firewalls.

Communication content filtering for protection against malicious down-loadable code or access to censored material is possible in our approach, but still not a practical concept. The reason for the latter is the possibility of defeating attempts to find information that is hidden through mechanisms, such as encryption, compression, or steganography ([Sch95, §1.2]).

A byproduct of our formalism is a new way in which audit is modeled. To the best of our knowledge there is no widely accepted model for audit. Audit records are represented as instantiations of tokens that have a certain color (data type). HCPNs can model regulated flows of information, in particular audit data. Modeling audit is therefore integrated with modeling the other parts of a firewall system with HCPNs.

The strong typing mandated by the formalism enforces rigor about assumptions, such as what information is present and usable at which places in the architecture. Furthermore, strong typing ensures unambiguous interface specifications for the combination of firewall mechanisms.

9.2 Future Work

The reference model in chapter 5 is limited to a single network policy domain. An avenue for future research is the development of a model that allows us to argue about the distribution and relationship of security policies and security services across several network domains.

Section 6.5 on properties and formal analysis of Colored Petri Nets is a limited investigation of what formal analysis can achieve and how firewall representations can be analyzed. We consider the following approaches promising for future research:

- A further investigation of the question of which desirable properties of firewall systems can be expressed as dynamic properties, which in turn can be verified mechanically for HCPNs. We conjecture that concentrating on invariants as an analysis technique is likely to be a rewarding strategy. The use of invariants avoids the problem of state explosion in occurrence graphs. Some of the properties that can be verified would allow designers to gain additional confidence into the firewall system under investigation.

- A search for behavior-preserving net reductions to avoid the state explosion of occurrence graphs. Such a technique combined with the increased computing power of future generations of hardware could be sufficient to make dynamic analysis of HCPNs using occurrence graphs a computationally tractable option for analysis. The same benefits as in the previous point apply.

- A search for more powerful tool support for the creation of a library of firewall mechanisms, the simulation of competing designs, and automatic verification of certain properties. These components would give designers the capability to explore competing designs in an interactive fashion, which could result in decreased development times and costs.

- An investigation of the question if this approach can become the basis for the generation of executable "firewall code" that fulfills efficiency requirements. The compilation of high-level descriptions into executable firewalls would have many of the same advantages and disadvantages that fourth generation programming languages have compared to first generation programming languages. There are potential advantages in respect to criteria, such as correctness, portability, maintainability, or productivity, and disadvantages in respect to efficiency.

- An investigation of issues in testing firewall components and systems modeled by HCPNs. An understanding of the extent to which testing can provide assurance about security critical questions is potentially beneficial to the confidence in the modeled system.

We consider the approach of automatic translation of a high-level description of the firewall into firewall configurations another promising area for future research. We expect a general high-level language for expressing security policies together with its automated translation into low-level representations, such as packet filter rules that drive the security devices, could have a positive impact on practical computer security. It would abandon manual mechanisms that are prone to error and enable the automatic translation of security policies to a variety of enforcement mechanisms.

9.3 Summary of Main Contributions

- We introduced a framework for the following contributions in the form of a waterfall model for the firewall life cycle.

- We introduced and described a reference model for firewall technology.

- We introduced a formalism for expressing the functionality of firewall mechanisms and firewall systems. The formalism is based on Hierarchical Colored Petri Nets. It can be used for graphical representation, simulation, and formal analysis.

- We described the generalized concept of authenticated signaling.

- We reported on the design, implementation, and exploration of a realization of authenticated signaling for ATM signaling.

9.4 Conclusions

This dissertation has advanced knowledge in firewall technology in several respects. It has introduced a reference model for firewall technology that has been applied during the design of a firewall security mechanism for ATM networks. We have demonstrated how the formalism of Hierarchical Colored Petri Nets can be used to describe, compose, simulate, and analyze firewall and mechanisms that are used to construct them.

Our prototype of authenticated signaling in ATM has shown that the signaling channel of connection management protocols can be used to secure connection management protocol messages and to provide security services as a basic building block for other components on the network, in particular firewall systems. The prototype demonstrated that firewall technology is a viable approach to network access control in high performance connection-oriented networks.

BIBLIOGRAPHY

BIBLIOGRAPHY

[ABLP91] Martín Abadi, Michael Burrows, Butler W. Lampson, and G. Plotkin. A Calculus for Access Control in Distributed Systems. Technical Report DEC/SRC-070, Digital Equipment Corporation (DEC), February 1991.

[AMP96] Ashar Aziz, Tom Markson, and Hemma Prafullchandra. *Simple Key-Management For Internet Protocols (SKIP).* Internet Engineering Task Force, Reston, Virginia, August 1996. Internet Draft (work in progress).

[AN94] Martín Abadi and Roger Needham. Prudent Engineering Practice for Cryptographic Protocols. *IEEE Computer Society Symposium on Research in Security and Privacy*, pages 122–136, May 1994.

[AR94a] Frederick M. Avolio and Marcus J. Ranum. A Network Perimeter with Secure External Access. In *2nd Symposium on Network and Distributed System Security (NDSS)*, San Diego, California, February 1994. Internet Society (ISOC).

[AR94b] Frederick M. Avolio and Marcus J. Ranum. A Toolkit and Methods for Internet Firewalls. In *Technical Summer Conference*, pages 37–44, Boston, Massachusetts, June 1994. USENIX.

[Atk95a] Randall Atkinson. *RFC-1825 Security Architecture for the Internet Protocol*. Network Working Group, August 1995.

[Atk95b] Randall Atkinson. *RFC-1826 IP Authentication Header*. Network Working Group, August 1995.

[Atk95c] Randall Atkinson. *RFC-1827 IP Encapsulating Security Payload (ESP).* Network Working Group, August 1995.

[ATM94] ATM Forum. *ATM User-Network Interface Specification, Version 3.1.* Prentice-Hall, Englewood Cliffs, New Jersey, September 1994.

[ATM96] ATM Forum. *ATM User-Network Interface Specification, Version 4.0.* ATM Forum, April 1996.

[BAN89] Michael Burrows, Martín Abadi, and Roger Needham. A Logic of Authentication. Technical Report SRC-039, Digital Equipment Corporation (DEC), February 1989.

[BAN90] Michael Burrows, Martín Abadi, and Roger Needham. Rejoinder to Nessett. *Operating Systems Review*, 24(2):39–40, April 1990.

[BC93] H. Bachatène and J. M. Couvreur. Reference Model for Modular Colored Petri Nets. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, volume 2 (of 5), Le Touquet, France, October 1993.

[BC94] Steven M. Bellovin and William R. Cheswick. *Firewalls and Internet Security.* Addison-Wesley Publishing Company, Inc., 1994.

[BCCH94] Bob Braden (editor), David Clark, Steve Crocker, and Christian Huitema. *RFC-1636 Report of IAB Workshop on Security in the Internet Architecture.* Network Working Group, June 1994.

[Bel95] Bellcore. *Q.port Portable ATM Signaling Software.* Bellcore, Bell Communications Research, Piscataway, New Jersey, February 1995.

[BGP$^+$94] Mary L. Bailey, Burra Gopal, Michael A. Pagls, Larry L. Peterson, and Prasenjit Sarkar. PathFinder: A Pattern-Based Packet Classifier. In *Proceedings of the 1st Symposium on Operating System Design and Implementation (OSDI)*, Monterey, California, November 1994. USENIX.

[BK85] W. E. Boebert and R. Y. Kain. A Practical Alternative to Hierarchical Integrity Policies. In *Proceedings 8th National Computer Security Conference*, Gaithersburg, Maryland, September 1985.

[Bro95] Frederick P. Brooks, Jr. *The Mythical Man-Month.* Addison-Wesley Publishing Company, Inc., second edition, 1995.

[Cal96] Christopher J. Calabrese. A Tool for Building Firewall-Router Configurations. *The USENIX Association, Computing Systems*, 9(3):239–253, Summer 1996.

[CCI88] CCITT. *Recommendation X-509 The Directory Authentication Framework.* CCITT, 1988.

[CER91] CERT. *Active Internet tftp Attacks, CA-91:18.* Computer Emergency Response Team, Carnegie Mellon University, Pittsburgh, Pennsylvania, September 1991.

[CER95] CERT. *IP Spoofing Attacks and Hijacked Terminal Connections, CA-95:01.* Computer Emergency Response Team, Carnegie Mellon University, Pittsburgh, Pennsylvania, January 1995.

[CER96] CERT. *TCP SYN Flooding and IP Spoofing Attacks, CA-96:21.* Computer Emergency Response Team, Carnegie Mellon University, Pittsburgh, Pennsylvania, September 1996.

[Cha92] D. Brent Chapman. Network (In)Security Through IP Packet Filtering. In *Proceedings of the 3^{rd} USENIX UNIX Security Symposium*, Baltimore, Maryland, September 1992. USENIX.

[Che92] William R. Cheswick. The Design of a Secure Internet Gateway. In *Proceedings of the 3^{rd} USENIX UNIX Security Symposium*, Baltimore, Maryland, September 1992. USENIX.

[Che97] Rudolph Chelminski. The Maginot Line. *Smithsonian Magazine*, pages 90–100, June 1997.

[Chu96] Shaw-Cheng Chuang. Securing ATM Networks. *Journal of Computer Security*, 4(4):289–329, 1996.

[CLR90] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, Massachusetts, 1990.

[Com95] Douglas E. Comer. *Internetworking with TCP/IP*. Prentice-Hall, Englewood Cliffs, New Jersey, third edition, 1995.

[Com96] Douglas E. Comer. IP over ATM: Concept and Practice. Interop talk on IP over ATM, March 1996.

[Cro82] David H. Crocker. *RFC-822 Standard for the Format of ARPA Internet Text Messages*. Dept. of Electrical Engineering, University of Delaware, Newark, Delaware, August 1982.

[CSI97] Computer Security Insitute CSI. 1997 CSI Firewall Product Analysis. `http://www.gocsi.com/firewall.htm`, 1997.

[CSV93] S. Castano, P. Samarati, and C. Villa. Verifying System Security Using Petri Nets. In *Proceedings of the 1993 IEEE International Carnahan Conference on Security Technology*, Ottawa, Ontario, Canada, 1993. IEEE.

[CZ95] D. Brent Chapman and Elizabeth D. Zwicky. *Building Internet Firewalls*. O'Reilley & Associates, Inc., Sebastopol, California, September 1995.

[DC93] Annette DeSchon and Danny Cohen. The ISI "Tunnel". Technical Report ISI/SR-93-358, Information Sciences Institute, University of Southern California, October 1993.

[Den82] Dorothy E. Denning. *Cryptography and Data Security*. Addison-Wesley Publishing Company, Inc., 1982.

[DH76] Whitfield Diffie and Martin Hellman. New Directions In Cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, November 1976.

[Dig92] Digital Equipment Corporation (DEC). *Screening External Access Link (SEAL) Introductory Guide*, 1992.

[Dij96] Barbara L. Dijker. *A Guide to Developing Computing Policy Documents.* The USENIX Association, 1996.

[DL94] Jacques J. Demäl and Alexander H. Levis. On Generating Variable Structure Architectures for Decision-Making Systems. *Information and Decision Technologies*, 19(4):233–255, 1994.

[DZ83] J. D. Day and H. Zimmermann. The OSI Reference Model. In *Proceedings of the IEEE*, volume 71, pages 1334–1340. IEEE, December 1983.

[Ebe92] Hans Eberle. A High-speed DES Implementation for Network Applications. In *Advances in Cryptology — Crypto '92*, pages 521–539. Springer-Verlag, August 1992.

[EF94] Kjeld Borch Egevang and Paul Francis. *RFC-1631 The IP Network Address Translator (NAT).* Network Working Group, May 1994.

[GJM91] Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering.* Prentice-Hall, Englewood Cliffs, New Jersey, 1991.

[GS96] Simson Garfinkel and Gene Spafford. *Practical UNIX & Internet Security.* O'Reilley & Associates, Inc., Sebastopol, California, second edition, 1996.

[HA94] Neil M. Haller and Randall Atkinson. *RFC-1704 On Internet Authentication.* Network Working Group, October 1994.

[Hal94] Neil M. Haller. The S/Key One-Time Password System. In *2^{nd} Symposium on Network and Distributed System Security (NDSS)*, pages 151–157, San Diego, California, February 1994. Internet Society (ISOC).

[Hil56] Tyrus Hillway. *Introduction to Research.* Boston, Houghton Mifflin, 1956.

[HJS91] Peter Huber, Kurt Jensen, and Robert M Shapiro. Hierarchies in Coloured Petri Nets. In G. Rozenberg, editor, *Advances in Petri Nets*, number 524 in Lecture Notes in Computer Science. Springer Verlag, 1991.

[How97] John D. Howard. *An Analysis Of Security Incidents On The Internet 1989-1995.* PhD thesis, Carnegie Mellon University, April 1997.

[HR92] Carlos A. Heuser and Gernot Richter. Constructs for Modeling Information Systems with Petri Nets. In K. Jensen, editor, *13^{th} International Conference on Application and Theory of Petri Nets*, number 616 in Lecture Notes in Computer Science, Sheffield, UK, 1992. Springer Verlag.

[Hug96] James Hughes. A High Speed Firewall Architecture for ATM/OC-3c. (unpublished), February 1996.

[ISV95] David Icove, Karl Seger, and William VonStorch. *Computer Crime*. O'Reilley & Associates, Inc., Sebastopol, California, 1995.

[IV97] Wesley Irish and Mark Verber. Private communication, 1997.

[Jen91] Kurt Jensen. Coloured Petri Nets: A High Level Language for System Design and Analysis. In G. Rozenberg, editor, *Advances in Petri Nets*, number 524 in Lecture Notes in Computer Science. Springer Verlag, 1991.

[Jen95] Kurt Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods, and Practical Use*, volume 2. Springer-Verlag, New York Inc., 1995.

[Jen96a] Kurt Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods, and Practical Use*, volume 1. Springer-Verlag, New York Inc., second edition, 1996.

[Jen96b] Kurt Jensen. *Design/CPN Overview of CPN ML Syntax. Version 3.0*, 1996.

[JK91] Ryszard Janicki and Maciej Koutny. Optimal Simulations, Nets, and Reachability Graphs. In G. Rozenberg, editor, *Advances in Petri Nets*, number 524 in Lecture Notes in Computer Science, pages 205–226. Springer Verlag, 1991.

[Jon95] Laurent Joncheray. A Simple Active Attack Against TCP. In *Proceedings of the 5th UNIX Security Symposium*, pages 7–19, Salt Lake City, Utah, June 1995. USENIX.

[KK94] David Koblas and Michelle R. Koblas. SOCKS. Socks package documentation, 1994.

[KPS95] Charles W. Kaufman, Radia Perlman, and Mike Speciner. *Network Security. Private Communication in a Public World.* Prentice-Hall, Englewood Cliffs, New Jersey, 1995.

[LABW92] Butler Lampson, Martín Abadi, Michael Burrows, and Edward Wobber. Authentication in Distributed Systems: Theory and Practice. Technical Report SRC-083, Digital Equipment Corporation (DEC), 1992. Reprinted in ACM Transactions on Computer Systems, volume 10, number 4, November 1992, 265–310.

[Lan81] Carl E. Landwehr. Formal Models for Computer Security. *ACM Computing Surveys*, 13(3):247–278, September 1981.

[Len95] Arjen K. Lenstra. *Documentation of LIP*. Bellcore, March 1995.

[LGL+96] Marcus Leech, Matt Ganis, Ying-Da Lee, Ron Kuris, David Koblas, and LaMont Jones. *RFC-1928 SOCKS Protocol Version 5.* Network Working Group, March 1996.

[Lie93] Armin Liebl. Authentication in Distributed Systems: A Bibliography. *ACM Operating Systems Review*, pages 31–41, October 1993.

[Lin93] John Linn. *RFC-1508 Generic Security Service Application Program Interface.* Network Working Group, September 1993.

[LR97] Maryline Laurent and Pierre Rolin. *Security Mechanisms within Control Plane.* ATM Forum, February 1997.

[LS87] Dennis Longley and Michael Shain. *Data & Computer Security. Dictionary of Standards, Concepts, and Terms.* Macmillan Publishers Ltd., 1987.

[LS96a] J. Bryan Lyles and Christoph L. Schuba. A Reference Model for Firewall Technology and its Implications for Connection Signaling. In *Open Signaling Workshop*, Columbia University, New York, New York, October 1996.

[LS96b] J. Bryan Lyles and Christoph L. Schuba. A Reference Model for Firewall Technology and its Implications for Connection Signaling. Technical Report CSD-TR-96-073, Department of Computer Sciences, Purdue University, West Lafayette, Indiana, December 1996.

[Lun89] Teresa F. Lunt. Access Control Policies: Some Unanswered Questions. *Computers & Security*, 8(1):43–54, February 1989.

[Lyl94] Bryan Lyles. Requirement for Authenticated Signaling, April 1994. ANSI Committee T1S1.5/94-118.

[Lyl95] J. Bryan Lyles. Private communication, 1995.

[Met93] Meta Software Corporation. *Design/CPN Reference Manual.* Cambridge, Massachusetts, 1993.

[Mil84] R. Milner. The Standard ML Core Language. Technical Report CSR-168-84, Edinburgh University Internal Report, 1984.

[Mil92] David L. Mills. *RFC-1305 Network Time Protocol (Version 3).* Network Working Group, March 1992.

[Mil96] David L. Mills. *RFC-2030 Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI.* Network Working Group, October 1996.

[Moc87a] Paul Mockapetris. *RFC-1034 Domain Names - Concepts and Facilities.* Network Working Group, November 1987.

[Moc87b] Paul Mockapetris. *RFC-1035 Domain Names - Implementation and Specification.* Network Working Group, November 1987.

[MS91] J. D. Moffet and M. S. Sloman. Content-dependent Access Control. *Operating Systems Reviews*, 25(2):63–70, April 1991.

[MS95] Perry Metzger and William Allen Simpson. *RFC-1828 IP Authentication using Keyed MD5.* Network Working Group, August 1995.

[MS96] Philip R. Moyer and E. Eugene Schultz. A Systematic Methodology for Firewall Penetration Testing. *Network Security*, pages 11–18, March 1996.

[MSR97] David M. Martin, Jr., Rajagopalan Sivaramakrishnan, and Aviel D. Rubin. Blocking Java Applets at the Firewall. In *5^{th} Symposium on Network and Distributed System Security (SNDSS)*, San Diego, California, February 1997. Internet Society (ISOC).

[MV92] William W. McLendon, Jr. and Richard F. Vidale. Analysis of an Ada System Using Coloured Petri Nets and Occurrence Graphs. In K. Jensen, editor, *13^{th} International Conference on Application and Theory of Petri Nets*, number 616 in Lecture Notes in Computer Science, pages 384–388, Sheffield, UK, 1992. Springer Verlag.

[NCS97a] National Computer Security Association NCSA. Firewall User Profile. An NCSA Focus Report. Carlisle, Pennsylvania, 1997.

[NCS97b] National Computer Security Association NCSA. NCSA Firewall Certification Program. `http://www.ncsa.com/fpfs/fwcert.html`, 1997.

[Nes90] Dan M. Nessett. A Critique of the Burrows, Abadi and Needham Logic. *Operating Systems Review*, 24(2):35–38, April 1990.

[NIS94] NIST. Digital Signature Standard (DSS). National Institute of Standards and Technology (NIST), May 1994. FIPS PUB 186.

[OL82] Susan S. Owicki and Leslie Lamport. Proving Liveness Properties of Concurrent Programs. *ACM Transactions on Programming Languages and Systems*, pages 455–495, July 1982.

[OOS$^+$97] Nick Ogurtsov, Hilarie Orman, Richard Schroeppel, Sean O'Malley, and Oliver Spatscheck. Experimental Results of Covert Channel Limitation in One-Way Communication Systems. In *5^{th} Symposium on Network and Distributed System Security (SNDSS)*, San Diego, California, February 1997. Internet Society (ISOC).

[Par93] Craig Partridge. *Gigabit Networking.* Addison-Wesley Publishing Company, Inc., 1993.

[Pet62] Carl A. Petri. Kommunikation mit Automaten. Technical Report 2 (Schriften des IIM), Institut für Instrumentelle Mathematik, Bonn, Germany, 1962.

[Pet81] James L. Peterson. *Petri Net Theory and the Modeling of Systems.* Prentice-Hall, Englewood Cliffs, New Jersey, 1981.

[Pic87] J. Picciotto. The Design of an Effective Auditing Subsystem. In *Symposium on Research in Security and Privacy*, Oakland, California, April 1987. IEEE.

[Pie96] Lyndon G. Pierson. Integrating End-to-End Encryption and Authentication Technology into Broadband Networks. Sandia National Laboratories, March 1996.

[Pnu77] Amir Pnueli. The Temporal Logic of Programs. In *18th Symposium on the Foundations of Computer Science*, pages 46–57, November 1977.

[Pos81a] Jon Postel, editor. *RFC-791 Internet Protocol.* Information Science Institute, University of Southern California, September 1981.

[Pos81b] Jon Postel, editor. *RFC-792 Internet Control Message Protocol.* Information Sciences Institute, University of Southern California, September 1981.

[Pos81c] Jon Postel, editor. *RFC-793 Transmission Control Protocol.* Information Sciences Institute, University of Southern California, September 1981.

[PR91] Helmut Plünnecke and Wolfgang Reisig. Bibliography on Petri Nets 1990. In G. Rozenberg, editor, *Advances in Petri Nets*, number 524 in Lecture Notes in Computer Science. Springer Verlag, 1991. Over 4000 references to publications dealing with Petri Nets.

[Pri97] Katherine E. Price. Misuse Detection Needs and Auditing System Capabilities (working title). Master's thesis, Department of Computer Sciences, Purdue University, West Lafayette, Indiana, December 1997. (to appear).

[PS91] Valerio O. Pinci and M. Shapiro, Robert. An Integrated Software Development Methodology Based on Hierarchical Colored Petri Nets. In G. Rozenberg, editor, *Advances in Petri Nets*, number 524 in Lecture Notes in Computer Science. Springer Verlag, 1991.

[PT97] Mohammad Peyravian and Thomas D. Tarman. Asynchronous Transfer Mode Security. *IEEE Network*, pages 34–40, May 1997.

[Ran92] Marcus J. Ranum. A Network Firewall. In *Proceedings of the 1st International Conference on Systems and Network Security and Management (SANS-I)*, June 1992.

[Ran93a] Karen T. Randall, editor. *Recommendation X-811 Information Technology - Open Systems Interconnection - Security Frameworks for Open Systems: Authentication Framework.* International Telecommunications Union, 1993.

[Ran93b] Marcus J. Ranum. Internet Firewalls — An Overview, October 1993. (unpublished).

[Ran93c] Marcus J. Ranum. Thinking About Firewalls. In *Proceedings of the 2nd International Conference on Systems and Network Security and Management (SANS-II)*, April 1993.

[Ran95] Karen T. Randall, editor. *Recommendation X-812 Information Technology - Open Systems Interconnection - Security Frameworks for Open Systems: Access Control.* International Telecommunications Union, 1995.

[Ran96] Marcus J. Ranum. Apparently OK Firewall — On the Topic of Firewall Testing. Ranum's opinions on the topic of firewall testing (unpublished), 1996.

[Ran97] Marcus J. Ranum. Do-It-Yourself Certificaiton Kit. *Computer Security Alert*, 171:1–8, June 1997.

[RLCB94] Marcus J. Ranum, Allen Leibowitz, Brent Chapman, and Brian Boyle. Firewalls-FAQ, 1994.

[Rub96] Aviel D. Rubin. Independent One-Time Passwords. *The USENIX Association, Computing Systems*, 9(1):15–27, Winter 1996.

[RW96] R. Martin Röscheisen and Terry Winograd. A Communication Agreement Framework for Access/Action Control. In *Proceedings of the Symposium on Research in Security and Privacy*, Oakland, California, May 1996. IEEE.

[SBHW95] Daniel Stevenson, Greg Byrd, Nathan Hillery, and D. Winkelstein. Design of a key-agile cryptographic system for OC-12c Rate ATM. In *Proceedings Symposium on Network and Distributed System Security.* Internet Society, February 1995.

[Sch95] Bruce Schneier. *Applied Cryptography.* John Wiley & Sons, Inc., second edition, 1995.

[Sch97] E. Eugene Schultz. When Firewalls Fail: Lessons Learned From Firewall Testing. *Network Security*, February 1997.

[Set90] Ravi Sethi. *Programming Languages. Concepts and Constructs.* Addison-Wesley Publishing Company, Inc., 1990.

[SH95] Karanjit Siyan and Chris Hare. *Internet firewalls and network security.* New Riders Pub., Indianapolis, Indiana, 1995.

[SHB95] Daniel Stevenson, Nathan Hillery, and Greg Byrd. Secure Communications in ATM Networks. *Communications of the ACM*, 38(2):45–52, February 1995.

[Shi95] Tsutomu Shimomura. IP Spoofing and Connection Hijacking. 3^{rd} Annual Workshop on Computer Misuse and Anomaly Detection (CMAD), January 1995. (presentation).

[Sil87] Robert D. Silverman. The Multiple Polynomial Quadratic Sieve. *Mathematics of Computation*, 48:329–339, 1987.

[SKK+97] Christoph L. Schuba, Ivan V. Krsul, Markus G. Kuhn, Eugene H. Spafford, Aurobindo Sundaram, and Diego Zamboni. Analysis of a Denial of Service Attack on TCP. In *Proceedings of the Symposium on Security and Privacy*, pages 208–223, Oakland, California, May 1997. IEEE.

[SKS95] Christoph L. Schuba, Berry Kercheval, and Eugene H. Spafford. Classical IP and ARP over ATM. Technical Report CSD-TR-94-024, Department of Computer Sciences, Purdue University, West Lafayette, Indiana, March 1995.

[Smi93] Martin Smith. *Commonsense Computer Security.* McGraw, second edition, 1993.

[SNS88] J. G. Steiner, C. Neuman, and Jeffrey I. Schiller. Kerberos: An Authentication Service for Open Network Systems. In *Proceedings, Winter USENIX*, Dallas, Texas, 1988.

[Spa88] Eugene H. Spafford. The Internet Worm Program: An Analysis. Technical Report CSD-TR-823, Department of Computer Sciences, Purdue University, West Lafayette, Indiana, 1988.

[SS94] Ted Smith and John Stidd. Requirements and Methodology for Authenticated Signaling, November 1994. ATM Forum 94–1213.

[SSK98] Christoph L. Schuba, Eugene H. Spafford, and Berry Kercheval. Prototyping Experiences with Classical IP and ARP over Signaled ATM Connections. *Journal of Systems and Software*, III 1998. (to appear).

[SSS+] Christoph L. Schuba, Eugene H. Spafford, Karyl F. Stein, Keith A. Watson, and Diego Zamboni. Firewall Evaluation Notes — Cisco Private Internet Exchange (PIX). (unpublished firewall product evaluation study).

[Sta95] William Stallings. *Network and Internetwork Security.* Prentice-Hall, Englewood Cliffs, New Jersey, 1995.

[Sti95] Douglas R. Stinson. *Cryptography — Theory and Practice.* CRC Press Inc., 1995.

[SV89] Manuel Silva and Robert Valette. Petri Nets and Flexible Manufacturing. In G. Rozenberg, editor, *Advances in Petri Nets*, Lecture Notes in Computer Science. Springer Verlag, 1989.

[Tal92] Anders Tallberg. The Property of Audit Trail. Technical Report C:252, Swedish School of Economics and Business Administration, 1992. `http://www.nan.shh.fi/NAN/Papers/AUTR92/autrtoc.htm`.

[Tar97a] Thomas D. Tarman, editor. *Phase I ATM Security Specification (Draft).* ATM Forum Technical Committee, July 1997.

[Tar97b] Thomas D. Tarman. Private communication, 1997.

[TPB+96] Thomas D. Tarman, Lyndon G. Pierson, Joseph P. Brenkosh, Barbara J. Jennings, Edward L. Witzke, and Marylou Brazee. Final Report for the Protocol Extensions for ATM Security Laboratory Directed Research and Development Project. Technical Report 96-0657, Sandia National Laboratories, March 1996.

[Tuc97] Allen B. Tucker, Jr. *The Computer Science and Engineering Handbook.* CRC Press Inc., 1997.

[Wik87] Åke Wikström. *Functional Programming Using Standard ML.* Prentice-Hall, Englewood Cliffs, New Jersey, 1987.

[Wir71] Niklaus Wirth. Program Development by Stepwise Refinement. *Communications of the ACM*, 14(4):221–227, April 1971.

[WL93a] Thomas Y. C. Woo and Simon S. Lam. A Framework for Distributed Authorization (Extended Abstract only). In *Proceedings of the Conference on Computer and Communications Security*, Fairfax, Virginia, November 1993. Association for Computing Machinery (ACM).

[WL93b] Thomas Y. C. Woo and Simon S. Lam. A Semantic Model For Authentication Protocols. In *Symposium on Research in Security and Privacy*, Oakland, California, May 1993. IEEE.

[WL93c] Thomas Y. C. Woo and Simon S. Lam. Authorization in Distributed Systems: A New Approach. *Journal of Computer Security*, 1993.

[Woo94] Charles Cresson Wood. *Information Security Policies Made Easy.* Baseline Software, Sausalito, California, 1994.

[Yah94] Raphael Yahalom. Secure Timeliness: On the Cost of Non-Synchronized Clocks. *The USENIX Association, Computing Systems*, 7(4):451–465, Fall 1994.

[You97] Michal Young. Private communication, 1997.

[YS96] Nicholas Yialelis and Morris Sloman. A Security Framework Supporting Domain-Based Access Control in Distributed Systems. In *4^{th} Symposium on Network and Distributed System Security (SNDSS)*, San Diego, California, February 1996. Internet Society (ISOC).

APPENDICES

Appendix A: Notation for Cryptographic Protocols

Principals participating in communication are denoted in lower case letters a or b. Principal a usually plays the role of the initiator (sender), principal b the acceptor (receiver) of a connection (data). If the role is not clear from the context the principals are additionally labeled with their role.

Messages that are transmitted in packets are denoted by msg. Received messages are labeled with a superscript asterix (*) to denote that the data might have been changed during transmission by an active wiretapper. Times are represented by t_i, where the subscript i is used to distinguish between different times. Numbers created by principal x are represented by n_x.

K is the symbol for encryption keys. If it is important whose principal's key it is, we will add the name of the principal as a subscript, e.g., K_a. K and K^{-1} are a public key pair with K^{-1} being the private key part. The same subscript rules apply. Encrypted messages are surrounded by curly braces, with the subscript stating the encryption key, e.g., $\{msg\}_{K_a^{-1}}$. Hash functions are abbreviated by $\mathcal{H}()$. $|$ denotes concatenation.

Appendix B: List of Acronyms

AAA	Authentication, Access control, Audit
AAAIE	AAA Information Element
AAL	Atm Adaptation Layer
ACF	Access (admission) Control Function
ACL	Access Control List
ADI	Admission/access Decision Information
AEF	Access Enforcement Function
AF	Authentication Function
AH	Authentication Header
AI	Authentication Information
API	Application Program Interface
ARP	Address Resolution Protocol
ATM	Asynchronous Transfer Mode
AudF	Audit Function
B-ICI	Broadband InterCarrier Interface
B-ISDN	Broadband Integrated Services Digital Network
BHLI	Broadband High Layer Information
BSD	Berkeley Software Distribution
CAM	Content Addressable Memory
CCITT	International Telegraph and Telephone Consultative Committee
CERT	Computer Emergency Response Team
CMA	Connection Management Agent
COAST	Computer Operations, Audit and Security Technology

CPN	Colored Petri Net
CRC	Cyclic Redundancy Check
CSI	Computer Security Institute
CSL	Computer Science Laboratory
CSMA/CD	Carrier Sense Multiple Access w/ Collision Detection
DLL	Data Link Layer
DMZ	DeMilitarized Zone network
DNS	Domain Name Service
DSS	Digital Signature Standard
DTE	Domain Type Enforcement
ESP	Encapsulating Security Payload
FCS	Frame Check Sum
GID	Group IDentifier
GSS	Generic Security Service
HCC	Host Call Control
HCPN	Hierarchical Colored Petri Net
HEC	Header Error Control
I/O	Input/Output
ICMP	Internet Control Message Protocol
IE	Information Element
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IF	Integrity Function
IP	Internet Protocol
IPC	Inter Process Communication
IPSEC	IP SECurity
ISDN	Integrated Services Digital Network
ISO	International Standards Organization
LAN	Local Area Network

LATM	Local area network ATM
LFSR	Linear Feedback Shift Register
LIS	Logical Ip Subnetwork
LLC	Logical Link Control
MAC	Medium Access Control
MAC	Message Authentication Code
MD5	Message Digest 5
MTU	Maximum Transfer Unit
NAT	Network Address Translation
NBMA	Non-Broadcast Multiple Access
NCSA	National Computer Security Association
NHRP	Next Hop Routing Protocol
NNI	Network Network Interface
NTP	Network Time Protocol
OAM	Operations And Maintenance
OC	Optical Carrier
OSI	Open Systems Interconnection
PARC	Palo Alto Research Center
PARCNIC	PARC Network Interface Card
PDU	Protocol Data Unit
PID	Protocol Identifier
PM	Preventive Maintenance
PNNI	Private Network-to-Network Interface
PVC	Permanent Virtual Circuit
QoS	Quality of Service
RIP	Routing Information Protocol
ROLC	Routing Over Large Clouds
SA	Security Association
SAAL	Signaling ATM Adaptation Layer

SADT	Structured Analysis and Design Techniques
SAR	Segmentation and Reassembly
SCC	Switch Call Control
SDU	Service Data Unit
SKIP	Simple Key management protocol for IP
SNTP	Simple Network Time Protocol
SONET	Synchronous Optical NETwork
SPARC	Scalable Processor ARChitecture
SPI	Security Parameters Index
SUN	Stanford University Network
SVC	Switched Virtual Circuit
TCP	Transmission Control Protocol
TFTP	Trivial File Transfer Protocol
UDP	User Datagram Protocol
UID	User IDentifier
UNI	User-Network Interface
VC	Virtual Circuit
VCI	Virtual Circuit Identifier
VPI	Virtual Path Identifier
VPN	Virtual Private Networking
WAN	Wide Area Network
WATM	Wide area network ATM
WORM	Write Once Read Multiple

VITA

VITA

Christoph L. Schuba was born in Heidelberg, Germany in 1968. After serving for two years in the German Red Cross as an Emergency Medical Technician, he studied Mathematics and Management Information Systems at the Universität Heidelberg and the Universität Mannheim, Germany, where he received his Vordiplom in 1991. Schuba worked in several computer consulting positions in Germany and Great Britain.

As a Fulbright Scholar, Schuba enrolled in the Department of Computer Sciences at Purdue University, where he joined the COAST (Computer Operations, Audit and Security Technology) research group and earned the Master of Science in 1993. He received the Doctor of Philosophy in December 1997 under the direction of Professor Dr. Eugene H. Spafford. Schuba performed part of his dissertation research in absentia in the Computer Science Laboratory at Xerox Palo Alto Research Center (PARC).

Schuba's research interests are in computer system security, high performance networking and distributed systems.