A SYSTEM FOR SECURE USER-CONTROLLED ELECTRONIC TRANSACTIONS

PROEFSCHRIFT

ter verkrijging van
de graad van doctor aan de Universiteit Twente,
op gezag van de rector magnificus,
prof. dr. F.A. van Vught,
volgens besluit van het College voor Promoties
in het openbaar te verdedigen
op donderdag 21 augustus 1997 te 16.45 uur.

door
Arne Helme
geboren op 19 augustus 1966
te Oslo

Het proefschrift is goedgekeurd door
prof. dr. S.J. Mullender, promotor.

A System for Secure User-controlled Electronic Transactions

Ph. D. Dissertation

by

Arne Helme

University of Twente

This dissertation was typeset with the LaTeX and AMS-LaTeX macro packages for TeX. The figures were drawn using Tgif and METAPOST. The dissertation was typeset at Shaker Publishing B.V. with a Linotronic 630 phototypesetter. The cover for the dissertation was designed by the author and based on artistic material created by Richard de Voer. The dissertation was in its entirety written and typeset using NetBSD, a free Unix-like operating system.

ISBN 90-423-0011-6

Shaker Publishing B.V.
St. Maartenslaan 26
6221 AX Maastricht
tel: 043 - 3260500
fax: 043 - 3255090
http://www.shaker.nl

Abstract

This dissertation is concerned with how security functions can be integrated into a personal computer to provide a subsystem that cannot be easily compromised by untrusted applications executing on that computer. The dissertation presents an overall system architecture for secure user-controlled execution of electronic transactions. The proposed system architecture is particularly suitable for personal computers hosting applications with conflicting security requirements.

In current open-networks transaction systems, cryptographic functions and protocols are used to protect the integrity of the data exchange. However, the ways in which the transaction services with security requirements are presented to the human users are often unprotected. In particular, many systems lack functionality to protect the integrity of the user interface. Since human users constitute the true ends of electronic-commerce transactions, the security risk is that human users may be lured into authorising dubious transactions.

This dissertation argues that the security of end systems hosting electronic-commerce applications depends crucially on the presence of a trusted path between the human user of the system and the signing keys used to authorise statements on behalf of the user. One way to achieve this, as is argued in the dissertation, is to embed the input and output devices of the human-visible computer interface into the trusted computing base of the system and protect the integrity of the user-interface elements presented to the user.

Foreword

This dissertation contains the results of almost five years of research in the field of computer and network security. The research was conducted as part of the Trust project at the University of Twente in the Netherlands.

During these years numerous people have helped me in different ways. In the following, I would like to express my gratitude to some of them who helped me the most.

First, I would like to thank my advisor, Sape J. Mullender for his support, help, patience, enthusiasm, and for having created a stimulating research environment at the Huygens laboratory.

A number of people helped me with proofreading the dissertation. In particular, my colleagues Peter Bosch, Feico Dillema, Erwin van Eijk and Tage Stabell-Kulø provided me with valuable comments. This year, Tage and I have been working together for ten years, and I hope that the next ten years will be even more successful.

I especially would like to express my gratitude to Feico, who helped me tremendously with improving the presentation of my work. Feico also deserves gratitude for introducing me to many local Dutch habits. He once expressed this very well himself: "Arne dragged me into science and I dragged Arne into town". Say no more...

During my stay in the Netherlands, Sylvia Hauenschild and Richard de Voer have become close friends of me. Richard also created the artwork that I used on the cover of the dissertation.

Last, but most important, I would like to thank my partner in life, Cindy van Lenthe, for believing in me and for being supportive even at those moments when working on the dissertation was all I did. Finally, I would like to thank both my own family and Cindy's family for all their help and support.

Arne Helme
Enschede, August 1997

Contents

—1—

Introduction

Over the past few years, the use of personal computers and public networks has experienced exponential growth, and this growth has created a market for electronic commerce. Using technologies such as the Internet and World Wide Web (WWW), users are now in a position to engage in transactions and contract negotiations with merchants electronically.

The use of personal computers as end systems for electronic-commerce applications is not entirely without risks. The problem is that many personal computers host applications with different security requirements. However, existing systems do not provide adequate security mechanisms to distinguish between applications with explicit security requirements and applications with no security requirements.

A particular threat is that a compromised end system can be used by an attacker to obtain authorisation statements from users without their consent. Such attacks can be launched by either fraudulent service providers or malicious users with access to the public networks.

In a security context, however, a personal computer can be viewed as an important resource. If such a computer is designed with security mechanisms built into its architecture in such way that it can detect attacks of the kind mentioned above, it becomes a powerful tool in realizing secure electronic commerce.

This dissertation is concerned with the design of a system that can be used for secure electronic-commerce applications. In this dissertation, an architecture of an end system for secure user-controlled electronic transactions is described together with a set of supporting applications.

This chapter is structured as follows. First, an overview with relevant background information on issues in electronic commerce is presented. The next section discusses human errors and factors in human-computer interface design. The third section states the hypothesis of the dissertation. Following the hypothesis is an overview of the research presented in this dissertation together with related work and a note on secure systems engineering. In the last section, an outline of the remaining chapters of this dissertation is presented.

1.1. Background

In the near future it will be common to own and use more than one personal computer. These will range from conventional personal computers (PCs) to miniature models, such as electronic purses and personal digital assistants. Ubiquitous

wireless computer networks will provide infrastructure for global communication in a similar way that telephone networks do for cellular telephones today.

One consequence is that more personal information will be stored and exchanged electronically, and that computers will be used to perform more tasks in everyday life. In addition, computers will be used to perform tasks on behalf of their owners at new locations, e.g., in banks and railway stations, and for new types of applications, e.g., electronic commerce.

In electronic-commerce settings, users and merchants will use transaction protocols to exchange information and contracts, e.g, representing payments or contracts in the usual legal sense. Personal computers will, to a larger extent, host commerce applications and be used as end systems of these transactions.

Computer security is complex, and most users have a poor understanding of the issues involved, and particularly poor understanding of the underlying security issues. Users react to information presented to them through the visible human-computer interface. If information presented through this interface is wrong, or if it has been tampered with, users may take actions that compromise their security. In a compromised system, the attacker may operate the user interface the way the user expects it to, leaving the user unaware of the actual actions performed by the compromised system. In this dissertation this problem, which is a fundamental one, will be denoted the user-interface integrity problem.

New technologies designed to make life easier are not entirely without problems. Since these technologies, e.g., personal computers, are used to carry out tasks of everyday life, information about daily tasks are automatically recorded by these devices. Much of this information is of a personal nature. Private information may be recorded and used without the consent of the owner—simply because it is not clear that this information is recorded and used in the first place.

Electronic commerce is an arena in which personal computers will be used by their owners to conduct transactions of a financial nature. Currently there is a growing market for electronic commerce, and most large players are more than willing to promote new technologies for their customers at low introductory fees. The introduction of smart cards by financial institutions in the Netherlands is a specific example of how this is done. The Internet is also a promising for electronic commerce purposes. The larger credit card companies have already devised system architectures for electronic payments in Internet environments.

Individual users may participate in electronic commerce using their home computers, interactive televisions sets, or by presenting electronic tokens issued by financial institutions. When conducting an electronic transaction, users respond to the information presented via the human-computer interface of the system. These user interfaces are typically operated by client-system applications of electronic commerce systems.

Although the way in which security mechanisms are used to preserve the integrity of the actual electronic transaction is usually quite sound, the way in which services with security requirements are presented to humans can be criticized. In particular, many systems lack functionality to preserve the integrity of the user

interface, and in the case of errors, provide users with little, if anything, to prove the events that took place. Since human users constitute the real end-points of electronic commerce, the security risk is that human user may be lured into authorising dubious transactions.

1.2. Issues in electronic commerce

Electronic systems whose security is based on cryptographic techniques are becoming common in many applications. Automatic Teller Machine (ATM) use cryptography for authentication and cellular phones use smart cards to authenticate callers for charging purposes. Global computer networks are used for electronic commerce. The integrity of these systems is preserved by cryptographic techniques.

Most people today carry out ATM transactions. In these transactions, the ATM card, together with a secret Personal Identification Number (PIN), is used as a mechanism for authenticating the user to the system. Many of these systems have been criticized severely as not being secure [Anderson, 1994b].

An important aspect of the problem is that most ATM cards use a magnetic stripe to carry the authenticating information intended for the bank's system. This allows cards being copied very easily. One of the most important technical problems with ATM cards and credit cards is that unauthorized third parties can easily obtain access to the card's information.

A solution proposed earlier was that smart cards should have a private display and a key pad [Abadi et al., 1990]. The smart card would show on its private display the amount to be authorized and the user would type the PIN on the smart card's private key pad. Given its small display, such smart cards cannot be used for anything but simple payments.

In current open-networks transaction systems, cryptographic functions and protocols are used to protect the integrity of the data exchange. However, the ways in which the transaction services with security requirements are presented to the human users are often unprotected. For example, a particular weakness is that the user of an ATM card or a smart card depends, for the correct execution of a transaction, on the party with whom the transaction takes place. This asymmetric level of trust serves only the protection of the merchant. In addition, the PIN used by the owner of the card to authorize a transaction must often be issued via an untrusted channel under control of the remote party. If a personal (networked) computer is being used, the transaction software may have been provided (on-line) by the remote party. There is a risk that a remote party can obtain an authorised statement from the user, and that the user has no way to deny this statement.

Cryptographic techniques can be used to establish secure channels, bind keys to principals, or verify that a piece of information was signed by a specific principal. This, however, is not sufficient to establish secure communication between a user and a remote service. When the *ends* of a communication channel are not properly defined, e.g., when a user executes uncertified third party software on a personal computer while at the same time carrying out electronic transactions with a remote

service, this is becoming a problem. The problem is that it is not possible to guarantee the integrity of the electronic transaction system. The application program or the operating system can be compromised, and thereby also compromise the electronic transaction system.

Consider the attack on distributed file systems using IP spoofing reported in [Brewer et al., 1995]. A client machine downloads executable files from a file server. Unless there is a trusted path from the client machine to the executable, it can be compromised during download. The reported attack is used to patch files on the fly while these are in transfer from the file server to the memory on the client machine. The attack works by forging a reply from the file server to the client machine. If the forged reply reaches the client machine before the legitimate reply, it will be accepted by the client machine instead of the legitimate reply. The attack has been used to patch the executable of the Netscape WWW browser to disable its security features, and to defeat Kerberos [Steiner et al., 1988] by attacking the *kinit* program. It highlights the importance of identifying the ends of communication. The trusted path to a file may extend well beyond the file server as in this example.

Even with correctly functioning applications, attacks can be launched against the human user. Consider the WWW spoofing attack reported in [Felten et al., 1996]. With existing WWW browsers, an attacker can mislead a victim to make inappropriate security-relevant decisions. These kinds of attacks are based on that with the existing infrastructures it is relatively easy for attackers to create a false but convincing world around the victim, and then trick the victim into doing something that would be appropriate if the false world were real. Electronic-commerce applications are particularly attractive to attack in this way.

Human users often rely on a context before they make a decision, e.g., that an ATM machine is bolted to the wall of their bank, or that a WWW page contain some recognisable information, e.g., a company logo. At some point during the communication, the victim authorises, e.g., a payment, or types a password which is transmitted to the service provider.

The attack described in [Felten et al., 1996] is exploiting users by masquerading as the service provider. It works as follows: The attacker creates a convincing but false copy of the WWW, and tricks the user to access this WWW and not the real WWW. This is far easier than it sounds. All the attacker has to do, is to sit between the victim and the real WWW. Whenever the victim requests some information (e.g., a hypertext page) from the real WWW, the attacker requests it for him, monitors or modifies it, and forwards it to the victim. Thus, all it takes is to trick the victim to request pages via the attackers machine instead of from the real WWW. This can easily be achieved by having a reference to the spoofed WWW linked to a popular site on the real WWW.

The fact that an attacker can monitor information requested or sent by a victim can have serious consequences. However, in [Felten et al., 1996] it is being argued there are far more serious concerns. Some WWW browsers are capable of executing software downloaded from remote sites, e.g., applet programs and other helper applications. Since the attacker is capable of providing the victim with just anything, code to change the appearance of the victim's user interface can also

be provided by the attacker. The practical implication is that the human-visible user-interface that is rendered on the victim's display can be modified to suit the attacker's goal of tricking the victim to issue authorisations. In addition, all traces of spoofed operations, e.g., the current page's location address can be hidden.

Note that if the victim has already been tricked into the spoofed WWW, the built-in security functions may not have any effect. The reason is simply that when the victim believes he has established a secure end-to-end connection with a service on the real WWW, the reality is that a connection is established between the victim's machine and the attacker's machine instead. Consequently, the symbols representing secure mode of operation that are presented to the victim, are not reliable as indicators of the current security policy.

Current WWW browsers are not capable of handling the types of attacks previously mentioned and no long-term solutions currently exist. Short-term solutions, e.g., to disable all applet functionality in WWW browsers, and to pay careful attention to the page locations, will reduce the risk of such attacks. The issues previously described can be considered user-interface integrity problems. The question is, thus, how to ensure that the human user only authorises intended operations.

1.3. The human factor

"Human error" is often cited as the cause of accidents in safety-critical systems. Since the human and technology are working together to control the system, it is more appropriate to view this as caused in part by the breakdown in the communication between the human user and the computer system [Modugno et al., 1996].

Today's personal computers respond not only to commands issued by the human user, but also to commands originating from other computers via computer networks. Malicious intruders or programs may compromise the security of personal computers and thereby contribute to the breakdown in communication between the user and system by providing the user with erroneous information.

Existing electronic-commerce systems do not properly place human authority in the control path of the electronic transaction process. An intriguing problem is that a user can be asked to issue an authorisation statement on the wrong premises. This problem is particularly present in systems where users are required to insert their cards into systems not under their control. For instance, when making payments in shops, or when using ATM machines.

The problem would also exist in systems where users employ personal computers on which they run an electronic transaction application. The personal computer could be running compromised software, or reveal the user's secrets unintentionally. Consequently, the system image presented on the display is not guaranteed to reflect the current state of the electronic transaction system correctly.

When a computer is executing electronic-commerce applications on behalf of a human user, the system can be considered a joint human-computer controlled system. Joint human-computer controllers are composed of three different components: the human user, the computer, and the interface between them [Modugno

et al., 1996]. The correct functioning of the controller depends on the correct functioning of each of these three parts independently and together.

In electronic-commerce systems, a human user is using a computer to request services. The computer responds to commands originating from the user, but also to commands (messages) received from remote computers acting on behalf of merchants. The human-visible interface provides the human user with indications about the current state of the transaction process.

Studies of real world accidents have shown that there are three main factors that contribute to the breakdown of communication between the human user and the computer [Leveson, 1995, Modugno et al., 1996]:

(1) the user had incomplete knowledge about the task the computer carried out,
(2) the user-interface lacked the robustness to face an imperfect human user, or
(3) there were ambiguities in the human-computer interface.

While education and experience are the only factors to increase the level of knowledge among users, the level of robustness and non-ambiguity in user-interfaces can be increased through good engineering practice.

Two necessary properties of a joint human-computer controlled system are that it lacks ambiguity, and that it is robust. Lack of ambiguity ensures that the human user can correctly determine the state of the system. The robustness property ensures that the system responds appropriately even when the human user makes errors.

In [Neumann, 1995], a wide range of accidents related to the use of computers are described. Many of these cases indicate that most security and safety related problems are introduced at the human-computer interface. To reduce the risks of using computer systems, more care needs to be devoted to the human-visible machine interfaces.

Interface design should be considered a fundamental part of the overall system design. Superficially, explicit prompting and confirmation indicators can be helpful, with insistence on self-defining inputs and outputs where ambiguities might otherwise arise.

User interfaces should also be developed using "good" engineering techniques such as consistent use of abstractions, information hiding, and parametrisation. In [Neumann, 1995], the following observations related to human-machine interfaces are offered:

- Although systems, languages, and user interfaces have changed dramatically, and we have learned much from experience, similar problems continue to arise—often in new guises.
- Designers of human interfaces should spend much more time anticipating human foibles.
- Manual confirmations of unusual commands, cross checking, and making backups are ancient techniques, but still helpful.

In a window-system environment, a command can have totally different effects in the different windows. Unless the user pays careful attention to the window contents, a password, for example, can be typed in an insecure window, or in a window executing on a remote machine. The result is that the authorisation statement is exposed to anyone monitoring the network traffic.

From using a system and observing its behaviour, users form a mental model based on the *system image* [Norman, 1983, Newman and Lamming, 1995] presented by the system. This mental model is not based on knowledge of the internal structure of the system, but on observations of the system's external behaviour (e.g., audible and visible output). The system image is presented to the human user through the human-computer interface, and it is this system image that guides human users in understanding the current state of the system.

A system, on the other hand, must also model the user's behaviour. In [Finin, 1989], it is being argued that a user model is the knowledge about a user that either explicitly or implicitly, is used by the system to improve the interaction. The knowledge is instantiated in the system's user model, e.g., the representation of the user.

In [Williges, 1987], user models are split into two groups: *conceptual models* (cognitive process models, cognitive structure models, cognitive strategy models) which mainly focus on representation of cognitive processes, and *quantitative models* (behaviour models, ergonomic models, computer-based simulation models, static models) which deal with numerical representation of the user's execution.

Most users of personal computers will often do "what the system tells them to do". They rarely ask themselves if whether the system image presented to them makes sense. A personal computer system has a certain "authority" that lay users accept unquestioned.

In electronic-commerce systems, the system image presented to the users describes the current state of an electronic transaction. To minimize the risk that a user authorizes a transaction unintentionally, the system should guarantee that the image presented to the user actually reflects the state of the current transaction in progress.

1.4. Problem statement

This dissertation is concerned with how secure end systems that are used for electronic-commerce should be designed. The main focus is on the specification of a system architecture and the security functions needed to ensure the security of personal computers when these are used as end systems for electronic-commerce applications.

Safety is an important goal to achieve: A solution should keep the human user in the control loop of authorisation decisions, and prevent authorisations from taking place unless these are provided by the user explicitly.

1.4.1. Hypothesis. In this dissertation, the following two hypothesises are made:

(1) Security in electronic-commerce applications can be achieved by incorporating security functions in the human-visible computer interface interface. More specifically, the integrity of information exchanged between a user's personal computer and a remote party's computer cannot be determined fully by the user's computer only. The process must be carried out with the consent of the human user. Therefore, the user must be able to observe and verify transaction information as authentic before authorisations are given.
(2) Security strong enough to support substantial financial and legal electronic transactions on behalf of people cannot be supported by software mechanisms alone. Hardware support secure logging are essential components—in fact, no modifiable software components should be part of a user's trusted computing base.

1.4.2. Research activities. The problems investigated in this dissertation require insight into the aspects of cryptography, computer security, network security, application security, and methods for presenting security-related information to human users.

The main activity is to design a system for secure user-controlled execution of electronic transactions and conduct an analysis of its properties.

1.5. The Trust architecture

In this dissertation, the main focus is on redefining the ends in end-to-end secure communication and on devising a new system architecture for end systems hosting electronic-commerce applications.

Central to the research is the concept of personal computers that are trusted by their human users (their owner) to act on their behalf in conducting electronic transactions. A human user trusts his personal computer to act on his behalf when he operates it, and not when somebody else does. It has input and output devices directly attached to it, and the users can interact with it without communicating with other devices (public terminals, for example).

Figure 1.1 illustrates the proposed system architecture. In the architecture, a distinction is made between trusted and untrusted components. The Trusted Computing Base (TCB) is a certified and unmodifiable subcomponent of the system that is trusted by the human user to execute operations with security requirements. The untrusted subcomponents include the Operating System (OS) and Applications. The Human User is presented with information through the Human-visible Interface. This interface is under control by the TCB during critical operations such as authorisation and access control. Communication with the "Outside World" (including computer networks and remote computer systems) is performed by the untrusted system components. The Security Perimeter is drawn between the trusted and untrusted components and is where the trusted path is located.

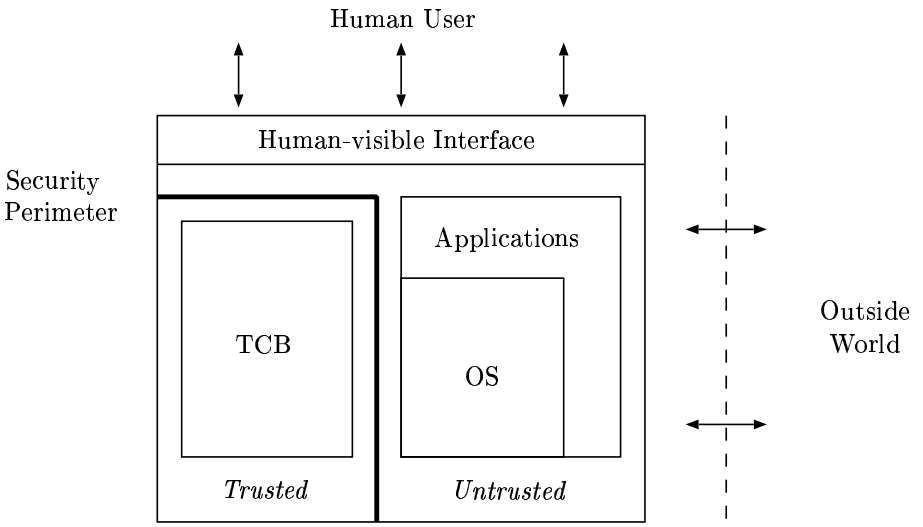

FIGURE 1.1. System architecture of a secure personal computer

Clearly, and in contrast to the communication channel that can be established between the two machines, the trusted path between a user and his Trusted Computing Base (TCB), cannot be based on the use of cryptographic functions. However, cryptographic functions can be used to provide underlying services to the mechanisms providing the communication channel.

User-interface integrity is essential for secure electronic commerce. Hardware support can be used to provide an integrity check of the display contents of the personal computer. In addition, by providing explicit signalling to the user via a trusted path that cannot be controlled by malicious software running on that computer, more confidence can be built in the integrity of the rendered information. Also, in use with a electronic transaction system, the display contents can be tied to the protocol and verified by the involved parties as what they agreed upon.

Secure and tamper-proof hardware plays an important rôle in electronic commerce, and in particular in the design of a secure user interface. In this dissertation, the purpose of secure hardware is to protect against online attacks via computer networks. A secure personal computer will contain some hardware security mechanisms that cannot be overridden by any malicious software running on that system.

Transaction protocols and logging functions are essential in electronic-commerce applications. Logging is essential when either system failures occur, or when disputes over transactions occur. Straightforward logging may not always be useful as evidence in a court of law. However, by associating logging with transactions, and by structuring transactions protocols with logging in mind, more confidence can be built in the contents of the logged information.

Authorisation and access control mechanisms are needed to prevent unauthorised access to system resources in general. A user may need to express that some system resources should be accessible to other users one or several times. Flexible authorisation mechanisms enable user to express complex access control patterns and delegations of authority between users.

Communication between computers is conducted via computer networks, and traditional methods such as cryptographic protocols are used to protect the communication. Network security is mainly concerned with mutual authentication of hosts (personal computers), and protection against man-in-the-middle attacks by providing secure communication channels at the network level. The provision of pairwise secure communication channels enable these hosts to exchange information issued by higher level applications securely between these computers.

The research described in this dissertation is mainly conducted as part of the Trust[1] project at the University of Twente. The main goal of this project is to study practical solutions to existing security problems in computer networks and electronic commerce systems. The central theme of this project has been personal computing and personal computers. It has been studied what impact this "personalisation" aspect has on the use of computers in networks. More specifically, it has been studied how remote services in electronic commerce systems can be accessed from personal computers and what implications this has for security.

The Trust approach is taken further in the Moby Dick project [Mullender et al., 1995], a joint European project (Esprit Long Term Research 20422) targeted at developing and defining the architecture of a new generation of mobile handheld computers. The project is a collaboration between Universities of Pisa in Italy, Twente in the Netherlands, and Tromsø in Norway.

In the Moby Dick project, a portable hand-held device with wireless communication facilities called the *personal digital companion* is used as the research vehicle, and the focus of the security activity is on a security model for sharing information between several personal computers belonging to one user, or between different users. The challenge is increased by the fact that the target environment constitutes low-bandwidth networks, and that some personal machines may often be off-line and unreachable.

1.6. Related work

The growing popularity of global information networks, such as the Internet, and information systems such as the WWW, has spawned much interest and research in computer security in general. Currently there is a broad consensus that the existing infrastructure is not very secure.

Security in computer networks is a widely researched topic (for an overview, see, e.g., [Voydock and Kent, 1983]. New network protocols (e.g., IP Next Generation [Hinden, 1996]) for global internet-working with security features included are being devised, and expected to be in use before the turn of the century.

[1]The Trust project is primarily sponsored by Rank Xerox EuroPARC, Cambridge.

End-to-end security in applications is subject to a growing interest. Many applications (e.g., WWW clients) already contain some security functions to authenticate registered services and establish secure communication channels with them. These efforts have resulted in several standards, e.g., Secure Socket Layer (SSL) [Freier et al., 1996]. During the last couple of years the computer industry has been one of the major driving forces behind these new standards.

Methods for authentication and delegation in distributed systems, as described in [Abadi et al., 1990, Lampson et al., 1992a], are also of interest since such techniques allow reasoning about trust and knowledge in the target environment. Logics of authentication and calculus for access control are fundamental for the establishment of trust in complex security systems.

Several designs of secure processors have been proposed. Dyad [Yee, 1994] is an example of a secure processor developed at Carnegie-Mellon University. The system runs the Mach [Accetta et al., 1986] operating system and uses special drivers to communicate with an insecure host computer via the system bus. The hardware is based on a Citadel coprocessor board [Palmer, 1992]

Legal aspects of computer crimes have been studied extensively in the context of ATM machine frauds [Anderson, 1994b, Anderson, 1994a, Anderson and Needham, 1995a], and the context of computer crimes (see, e.g., [Cheswick, 1992, Sterling, 1992]). The legal situation of using cryptography in civilian appliances are also extensively studied (see, e.g., [Anderson, 1995, Koops, 1996]). A summary of the U.S. crypto policy situation can be found in [Landau et al., 1994].

To guide the development of secure systems, secure systems evaluation criteria have been created. The Trusted Computer System Evaluation Criteria (TCSEC) [Department of Defense, 1985] (commonly referred to as the "Orange Book") and Canadian Trusted Computer Product Evaluation Criteria (CTCPEC) [Canadian System Security Centre, 1993] are two of the best known such criteria. The Information Technology Security Evaluation Criteria (ITSEC) [ITSEC, 1991], a European-developed criteria filling a rôle roughly equivalent to the TCSEC has been defined. The Common Criteria for Information Technology Security Evaluation (CCITSE) [TPEP, 1996] is a multinational effort to write a successor to the TCSEC and ITSEC that combines the best aspects of both.

1.7. Secure System Engineering

Although the advent of global information networks, such as the WWW, has spawned much interest into the fields of computer and network security, most efforts still go into the development of new features (e.g., as illustrated by the "battle" between the Netscape and Microsoft corporations over WWW browsers [Ramo, 1996]). As long as market forces rule, this scenario is not likely to change.

Ideally, the development of a new application with security requirements should follow a System Security Engineering (SSE) process iteratively (see e.g., [Weiss, 1991, Amoroso, 1994]) until the risks of its use is considered acceptably low by (in some cases, external) review.

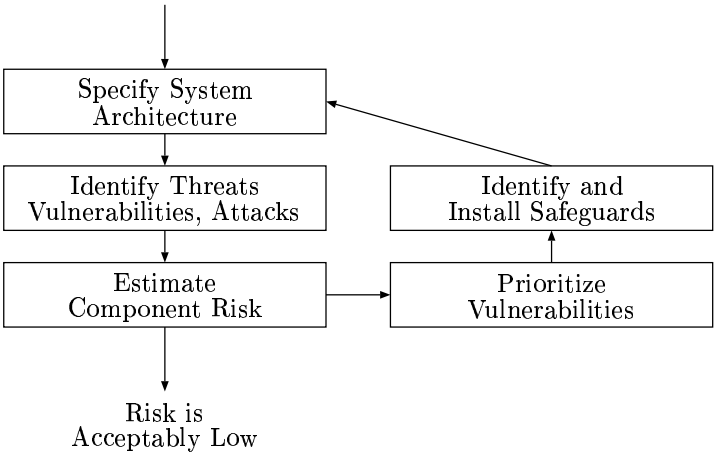

FIGURE 1.2. The System Security Engineering process (adapted from [Amoroso, 1994])

Figure 1.2 illustrates the SSE process. Initially, a system specification phase is conducted, and the architecture of the system is specified. Following the specification is a phase where threats, vulnerabilities and attacks on the specified architecture are identified.

Threats are identified using a threat tree approach where high-level potential threats are used as a starting point and further decomposed into lower level threats.

The next step is estimate component risks, and if these are found to be acceptably low the SSE process is terminated. Otherwise the vulnerabilities of the system are prioritised, and new safeguards are identified and installed. This, again, results in another iteration of the SSE process, and new risk estimates.

The goal of the research described in this dissertation is not to develop a new product or to devise a completely secure and foolproof system. In fact, completely secure and foolproof systems do not exist. Therefore, no such claims are made in this dissertation either.

The main activity of the research is focused on a subset of the SSE activities with focus on end-system security and the use of personal computers. However, the observations should be viewed as information that can be used as input to SSE-based design processes and development of secure electronic-commerce systems.

1.8. Thesis outline

Some of the chapters in this dissertation survey methods and define terms used throughout the rest of the dissertation. Other chapters describe design, implementation and analysis of the proposed architecture.

Chapter 2 surveys the field of cryptography. The survey does not aim at being a complete presentation of the field, but is instead focused on issues of relevance to the discussion in other chapters of the dissertation.

Chapter 3 is concerned with fundamentals of computer and network security. The focus is on security models and design methodologies underlying most secure computer systems. The chapter introduces the basics of computer security and defines the terms that are used in the remainder of this dissertation. It also serves the purpose of providing an understanding of computer security in general.

Chapter 4 outlines the architecture of an end system for electronic transaction processing that protects its users under the threat model described above. The architecture is called Trust, and describes a computer system that provides a trusted path between a user and his signing keys, and incorporates security functions into the human-visible computer interface. The architecture supports interaction between the user and a device that enables a transaction to be conducted securely between that device and another system. The transaction is not conducted by the device unless it is explicitly authorized by the user to do so.

Chapter 5 discusses the use of transaction protocols and logging facilities in electronic-commerce applications. The main focus is on protocol requirements and specification. An example protocol for contract signing is also presented.

Chapter 6 describes how authorisation and access control is achieved in μ-CAP. The μ-CAP architecture is designed to enable its users to define their own access control policies and have them enforced by remote servers. In μ-CAP, access rights can be propagated between users and exercised by the server without any communication with the object server prior to access requests.

Chapter 7 describes the design and implementation of a network-level security system, and how this solution protects communicating pairs of machines against active and passive attacks from a third party. The system is implemented as an extension to the Unix-like system, and integrated into the TCP/IP protocol suite.

Chapter 8 summarises the results of this dissertation and presents its conclusions. An overview of future work is also presented.

—2—

Applied Cryptography

This chapter surveys the field of cryptography and the use of cryptographic methods to increase the level of security in computer systems. The survey does not aim at being a complete presentation of the field, but is instead focused on issues relevant to the discussion in the remaining chapters of this dissertation.

The field of cryptography has a long history and cryptographic methods have been used since ancient times. Cryptographic research has been mainly conducted by military intelligence agencies, but during the last two decades academic institutions have also contributed largely to the field. Recent developments in network computing and electronic-commerce applications have spawned much interest into the field of cryptographic research.

Cryptography is the foundation of security in computer and network systems, and it is expected to play a significant rôle in the realization of electronic commerce. Modern cryptography is based on advances in mathematical number theory and complexity theory. Using mathematical transformation, such as permutations, substitutions and nonlinearity, information can be made diffuse and unreadable to anyone who does not have knowledge of the procedure of transformations being used.

In some countries, governments seek to limit individual privacy in electronic communication and disallow export and import of products that employ cryptographic methods. The legal situation of cryptography has implications for the realization of cryptographic products and integration of cryptographic methods into new application areas.

This chapter is structured as follows. First, an overview of cryptography and its goals are presented an the most fundamental concepts are described. Following the overview is a presentation of cryptographic methods including examples from the literature. The last section of the chapter is devoted to legal aspects of applying cryptographic methods in civilian applications.

2.1. The goals of cryptography

With the development of new application areas, the goals of cryptography also change. In general, cryptography provides methods that enable communicating parties to develop trust that their communication have the desired properties even in the presence of untrusted parties and adversaries. These properties include, but are probably not limited to the following [Rivest, 1990]:

Privacy: An adversary learns nothing useful about the messages being exchanged.

Authentication: The recipient of a message can convince himself that the message as received originated with the alleged sender.

Signatures: The recipient of a message can convince a third party that the message as received originated with the alleged signer.

Minimality: Nothing is communicated to other parties except that which is specifically desired to be communicated.

Simultaneous exchange: Something of value (for example a signature on a contract) is not released until something else of value (for example, the other party's signature) is received.

Coordination: In a multi-party situation, the parties are able to coordinate their activities toward a common goal even in the presence of adversaries.

Collaboration threshold: In a multi-party situation, the desired properties hold as long as the number of adversaries does not exceed a given threshold

2.2. Cryptographic methods

In this section, an introduction to the field of cryptography is presented. The field of cryptography is concerned with methods for communication in the presence of adversaries. Its history can be traced back to ancient times.

Much has been written about cryptography. In [Kahn, 1967], the history of cryptography and events of importance to the evolvement of the field and history in general are presented in great detail. Many excellent text books have been written about cryptographic methods and their applications in the field of secure communication and data security (see for example [Denning, 1982, Rivest, 1990, Simmons, 1992, Schneier, 1996]).

A general cryptographic method is to take an original (*plaintext*) message and transform it into an encoded version of the message (the *ciphertext*). A cryptosystem consisting of an encryption function and a decrypt function is used to transform the plaintext message into ciphertext and the ciphertext into plaintext again. Both the encryption and the decryption functions can be parametrised to facilitate the use of a *key*.

More specifically, the classical *secret-key cryptosystem* can be defined as follows [Rivest, 1990]:

- A *message space* $\mathcal{M}$: a set of strings (plaintext messages) over some alphabet.
- A *ciphertext space* $\mathcal{C}$: a set of strings (ciphertexts) over some alphabet.
- A *key space* $\mathcal{K}$: a set of strings (keys) over some alphabet.
- An *encryption function* E mapping $\mathcal{K} \times \mathcal{M}$ into $\mathcal{C}$.
- A *decryption function* D mapping $\mathcal{K} \times \mathcal{C}$ into $\mathcal{M}$.

The functions E and D must have the property that $D(K, E(K, M)) = M$ for all $K \in \mathcal{K}, M \in \mathcal{M}$.

Two parties who wish to use the cryptosystem agree on a key K which they will keep secret. Secure communication is achieved by transmitting the ciphertext $C = E(K, M)$. Since $M = D(K, C)$, the recipient can obtain the plaintext message by decrypting the ciphertext message using the key K.

If an adversary has knowledge of only the cryptographic method being used and a ciphertext message created with that method, he should not be able to reveal the plaintext message. The key with which a ciphertext message was created should be required to reveal the plaintext message again.

Public-key (asymmetric) cryptography [Diffie and Hellmann, 1976] is designed so that pairs of keys are used. Each user of a public-key cryptosystem creates two keys called the *public key* and the *private key* with the following properties:

- it is computationally infeasible to deduce the private key from the public key,
- anyone with knowledge of the public key can use it to encrypt a message, but not decrypt it,
- anyone (in most settings, only the owner) can use the private key to decrypt messages encrypted with the public key.

The public key K is made publicly available (for example, by printing in it a catalogue) and the private key K^{-1} is kept secret by the creator (owner). A plaintext message is encrypted using the function $E(x)$ and the public key, and decrypted again using the function $D(x)$ and the associated private key.

When someone *signs* a message M with a private key, anyone can *verify* the result with K and derive, from the assumptions that K is associated with a particular user and that this user keeps K^{-1} secret, and the properties of public-key cryptosystems mentioned above, that the user must have signed M. Anyone with this knowledge of K can verify that the private key belonging to the user in question once signed M. This use of sign and verify operations is the underlying principle of *digital signatures*.

Public-key systems are usually based on the use of *trapdoor one-way functions*. A trapdoor one-way function is a family of invertible functions f_z and indexed by z. Given z, it is easy to find two algorithms E_z and D_z that compute $f_z(x)$ and $f_z^{-1}(y)$ for all x and y in the domain and range of f_z respectively. However, it is computationally infeasible to compute $f_z^{-1}(y)$ with the knowledge of E_z only.

Many secret-key and public-key algorithms have been proposed in literature. Some of these are resistant to *cryptanalysis* while others are not. In the following, two of the more successful algorithms, the Data Encryption Standard (DES) secret key algorithms and the Rivest-Shamir-Adleman (RSA) public-key algorithm are described in more detail. Following the description of these two algorithms is also a discussion of adequate key-lengths for secret-key and public-key cryptosystems, and hash functions that are used to create message digests.

2.2.1. Data Encryption Standard. The Data Encryption Standard (DES) [National Bureau of Standards, 1978] secret-key algorithm has for almost 20 years been the standard against which other algorithms are compared. DES uses 56

bits keys and operates on 64-bit blocks of plaintext messages. Encryption and decryption is performed using the same algorithm (although the key is initiated different for encryption and decryption).

For each block of plaintext, the DES algorithm is constructed to carry of 16 rounds of substitutions and a permutations called the internal S-boxes of DES. A detailed description of the internal S-boxes used in DES are found in most textbooks on cryptography (e.g., [Schneier, 1996]).

DES can be operated in the following modes [National Bureau of Standards, 1980]: Electronic Codebook (ECB), Cipher Block Chaining (CBC), Output Feedback (OFB), and Cipher Feedback (CFB). These modes specify different ways to to combine the results of applying DES to blocks of plaintext messages. The ANSI standard specify ECB and CBC for encryption and CBC and CFB for authentication [ANSI X3.106, 1983].

The security of DES has been widely discussed and it is still a controversial topic. The controversy is primarily over the 56-bit key length, and the construction of the internal S-boxes of the DES algorithm.

When DES was first published, it was noted that a 56-bit would probably not be adequately large and that a cryptosystem that uses a 56-bit key-space would be vulnerable to brute-force attacks by searching the entire key-space (see, e.g., [Diffie and Hellmann, 1977]).

Much of the design criteria behind the internal S-boxes of DES have been kept secret, and the secrecy surrounding their design has raised concerns about whether hidden vulnerabilities were designed into the algorithm. Few weaknesses have been found, although it has been reported that certain structures have been found that appears to weaken the system [Hellman et al., 1976].

In addition to the concerns mentioned above, many attacks against DES have been reported. Using differential cryptanalysis [Biham and Shamir, 1991], a chosen-plaintext attack against the algorithm that is more efficient than a brute force attack. Linear cryptanalysis is another technique that has been used to attack DES [Matsui, 1994]. A more comprehensive list of attacks can be found in [Schneier, 1996].

The DES algorithm can be implemented relatively efficiently in hardware. In [Eberle, 1992], an implementation of a DES chip that can encrypt at a rate of 1 gigabit (16.8 million blocks) per second is presented. Software implementations of DES are orders of magnitudes slower, but modern personal computers can still encrypt at a data rate of several megabits per second.

In addition to DES, many other secret-key cryptosystems and attacks on them are described in literature, for example, the IDEA [Lai, 1991], RC5 [Rivest, 1995] and WAKE [Wheeler, 1993] algorithms.

2.2.2. Rivest-Shamir-Adleman. A well-known and widely used public-key cryptosystem is RSA [Rivest et al., 1978]. RSA is a block cipher in which both the plaintext, ciphertext and key alphabets are integers. Encryption and decryption are both achieved using modular exponentiation operations. The RSA cryptosystem is currently widely in use in cryptographic applications and products.

In order to use the RSA cryptosystem, each user creates two randomly chosen large prime numbers p and q. Using these two prime numbers the product $n = p * q$ and a randomly chosen encryption key e, are computed in such way that the latter is relative prime to $\phi(n) = (p-1)(q-1)$. The decryption key d is then computed by finding the multiplicative inverse of e modulo $\phi(n)$. A user of the RSA cryptosystem publishes the components n and e as the public key, and keeps the components d, p, q and $\phi(n)$ secret since these resemble the private key. After the public and private keys have been created, the components p and q can be discarded.

When using RSA, encryption of a plaintext message m into the ciphertext message c is achieved by computing $c = m^e (\text{mod } n)$. Similarly, decryption of a ciphertext message c to a plaintext message m is achieved by computing $m = c^d (\text{mod } n)$.

The strength of the RSA cryptosystem depends on the difficulty of the factorization of large prime numbers, and breaking RSA is estimated to be as hard as to factor n into its two prime number components p and q. This conjecture has never been mathematically proven.

Advances in factorization techniques is a concern for the security of RSA. The initial challenge put forward by RSA's inventors was 129-digit number (RSA-129). Using a large numbers of networked computers world-wide, a factorization of RSA-129 was achieved in 1994 [Atkins et al., 1995] using the quadratic sieve factoring method [Lenstra and Lenstra, 1993]. The next challenge, RSA-130 was factored in 1996 using an improved version of the method that was used for the factorization RSA-129 [Lenstra, 1996]. The latter required only a fraction of the computing resources used to factor RSA-129.

2.2.3. Key lengths. In addition to the strength of a cryptographic algorithm, the length of the keys used to encrypt information also plays an essential rôle as to what level of security that can be achieved. Obviously, if a key is too short, for example, then it is vulnerable to a brute-force attack from adversaries that search the entire key-space.

In a recent statement, a number of cryptographers and computer scientists gave their advice concerning the minimal key lengths to use for symmetric cryptosystems in order to provide an adequate level of commercial security [Blaze et al., 1996a]. Their advice is based on the fact that a modest increase in computational cost can produce a vast increase in security, and can be summarized as follows:

> "It is simplest, most prudent, and thus fundamentally most economical, to employ a uniformly high level of encryption: the strongest encryption required for any information that might be stored or transmitted by a secure system."

Further, they advocate that encryption keys employed in symmetric-key cryptographic systems should be at least 90 bits long, and that this will suffice for most applications for the next 20 years, even when the predicted advances in computing power are taken into account.

Adequate key lengths for public-key cryptosystems is a different story. RSA, for example, currently requires key lengths in the range of 768–1024 bits to be considered secure for applications with non-military security requirements.

It should be noted that the discussion on sufficient key-lengths is primarily of a political nature. There are no technological reasons why a cryptographic system should be "crippled" by the use of short encryption keys.

2.2.4. Hash functions. Hash functions are commonly used in conjunction with public-key cryptosystems to create digital signatures of plaintext messages. Since it is not cost effective to sign whole plaintext messages using public key encryption, the plaintext message is instead hashed to a short digest, and that digest is signed instead.

The usefulness of hashing messages to digests are two-fold. First, it is more efficient to sign a hash of a message instead of the message itself. Secondly, if it is hard to find two messages with the same message digest, the digest itself can be looked upon as a "fingerprint" of the message, and thus, a unique identifier.

Most hash functions view the input as a sequence of n-bit blocks. Each block is processed one at at time to produce an n-bit message digest typically in the range of 128–256 bits.

One recognized property a hash function should have is that of a one-way function [Diffie and Hellmann, 1976]. Thus, it should be computationally infeasible to invert a hash function and compute a message with a given digest. More formally, given a hash function h and a message digest Y, it should be computationally infeasible to find any message X such that $Y = h(X)$.

Hash functions need other security properties than just being one-way functions. One important property is *collision freedom*: a hash function h is considered collision free if it is computationally infeasible to find any two input messages X and Y such that $h(X) = h(Y)$. This property is of importance for digital signature schemes since, if h is not a collision free algorithm, it would be feasible to find a different message and substitute it for the original (authentic) message. Collision freedom is therefore considered an important property of hash functions, and hash functions that are collision free are stronger than those that are only one-way functions.

Another property that is related to collision freedom is *correlation freedom*. This property has been proven to be strictly stronger than collision freedom [Anderson, 1993].

Furthermore, correlation freedom is not the only property a good hash function should have. The properties of a good hash function depends heavily on the context in which it is employed and their real requirements can be summarized as follows:

- It is often assumed implicitly that a hash function has information hiding properties. In the situations where this is of importance, it is not sufficient that the hash function is collision free. A hash function should possess both local and global one-way properties. Otherwise it may leak bits unintentionally.

- A hash function should not interact with a given signature or authentication scheme in a way that can compromise its security. Since signature and authentication schemes are based on many different mathematical constructs, there are also many ways in which hashing may interact with them. For example, a good hash function can be used to break up the homomorphism property of RSA by passing the data through it first.
- Hash functions should be complementation, addition, and multiplication free. That is, it should not be feasible to find any input values X and Y such that $h(X) = \sim h(Y)$, where $\sim X$ is the binary complement of X. Further, it should be feasible to find inputs X, Y and Z such that $h(X) + h(Y) = h(Z)$, or $h(X)h(Y) = h(Z) (\text{mod } N)$. In the latter case the hash function h is multiplication-free mod N.

The use of a hash function for security purposes requires a good understanding of the properties of the hash function itself, but also a of the properties of the system in which it will be used.

One commonly used hash function is the MD5 message-digest algorithm [Rivest, 1992]. MD5 produces a fixed-length 128 bits message digest of arbitrary-length messages. MD5 is an extension of the MD4 [Rivest, 1991] message-digest algorithm. MD5 is currently used in many cryptographic applications, in particular, in combination with RSA.

A relatively new hash function is Tiger [Anderson and Biham, 1995]. The Tiger hash function outputs 192 bits message digests, but can also produce 128 bits output, and therefore be used as a drop-in replacement for existing implementations of other hash functions.

2.3. Cryptography and the law

An overview of applied cryptography would not be complete without a discussion of the legal situation of applying cryptographic techniques in civilian applications. In this section, the rôle of nonrepudiation, digital signatures, personal use of cryptography, and export control restrictions are discussed.

The current crypto policy situation in Europe is rather diverse [Anderson, 1995, Koops, 1996]. Few of the countries enforce the same policies on the use of cryptography, and the legal status of cryptographic evidence varies between countries. Some countries have laws against personal use of cryptography, and enforce export restrictions on cryptographic products to prevent dissemination of these technologies to other nations.

The majority of fielded crypto applications are not concerned with message secrecy but with authenticity and integrity. Their goal is to assure that communicating parties can be ascertain their mutual identities, and that the integrity of messages transmitted over insecure networks can be guaranteed. These services are fundamental for electronic commerce in global networks.

2.3.1. Digital signatures. The use of digital signatures is a concern for electronic commerce. When someone denies having sent a message (nonrepudiation),

information about the events that took place is essential for an arbitration to take place.

Authorities, certificates and digital signatures are central to the use of cryptography in electronic commerce, and in particular, to nonrepudiation. Indeed, public-key cryptography is considered one of the cornerstones of electronic commerce and much of the promise of electronic commerce is based on the assumption that public-key crypto systems are hard to break.

Certificates are issued by authorities to provide bindings between proper names of principals and their public keys. Digital signatures are used to create signed statements in such way that the private key vouches for the contents of the signed message. This service is also used to for nonrepudiation purposes. A digital signature is typically valid inside a domain where an authority has issued a certificate for the associated public key.

Currently public-key certification infrastructures are slowly emerging. However, they are not emerging as one certification hierarchy. Many large corporations, e.g., the major credit card companies, build their own certification domains. The consequences are that users must maintain keys registered with multiple organisations, and that each signing key will authorise different actions in different domains, or be valid in one domain only.

In [Kent, 1996], it is being argued that companies that operate generic certification services must balance liability concerns, acceptable cost models, levels of authentication assurance, and name space issues. This is also the main reason why certificates are expected to be valid, mainly, in their issued domain only.

The legal status of digital signatures is in most places still uncertain. Few countries, if any, have given the same legal status to digital signatures as to ordinary handwritten signatures. However, the acceptance of digital signatures as evidence in court is a prerequisite for the success of electronic commerce. Without such acceptance, the liability aspect of electronic transactions is also uncertain. Since public-key signatures are deemed to degrade over time, it is also important to have legal legislations governing the time frames in which electronic signatures should be deemed valid.

2.3.2. Personal use of crypto. In some countries, governments seek to limit individual privacy in electronic communication (see, e.g., [Federal Bureau of Investigation, 1992] and the discussions in [Denning, 1993] and [Anderson, 1995]). This is done through legal legislations restricting how products employing strong cryptography can be used, and by providing cryptographic products with built-in law enforcement wiretapping functions.

Key escrow encryption systems [Denning and Branstad, 1996] are being advocated as a solution to the problem of conflicting interests between an individual user's need for privacy, and the right of law enforcement agencies to wiretap (with an appropriate court order, one hopes). However, such approaches are not entirely without any risks [Abelson et al., 1997], and the design and implementation of secure cryptographic devices with wiretapping functions are prone to errors (see, for example, [Blaze, 1994b]).

When key escrow is used to provide law enforcement with wiretap capabilities of encrypted communication, the question that arises is how the key management of these systems is done. An interesting topic is the decision of who should have access to the keys and under what conditions keys should be disclosed to law enforcement agencies.

In some countries, it has been argued that a court order should be obtained and presented by law enforcement agencies before a particular key is disclosed. It has also been claimed that this would be sufficient for most democratic countries. However, policies tend to change with the politicians.

In Norway, for example, a recent investigation revealed that during the last 40 years, law enforcement agencies were used extensively by politicians for politically motivated purposes to monitor the activities of both individuals and organisations with different political views [Lund, 1996]. Sometimes wiretaps were conducted by law enforcement agencies without even applying for a court order. In many cases where a court order was applied for, an approval was given with no questions asked about the relevance of the wiretap. Norwegian law enforcement and military intelligence agencies are currently under strong public scrutiny.

Note that if users must disclose their private keys to a central authority before this authority signs a certificate for their public counterpart, it also creates a liability problem with implications for electronic commerce. In these situations the authority in question can, at any time, sign statements on behalf of the key-owner, and the owner has no way to deny these statements. Thus, such use of central authorities will weaken the effect of applying cryptography in electronic commerce systems.

A more serious implication is that evidence can be constructed, and used against the key owner. In systems where the use of trusted third parties (TTP) is involved, the underlying assumption about their trustworthiness, does not remove the inherent risk that such events could occur.

2.3.3. Export control. Some countries disallow export and import of products that employ cryptography. In the United States, for example, cryptographic products are currently subject to the same import and export regulations as arms and munitions. Export licences can be obtained for signature-only products, and for products that employ symmetric-key encryption with reduced key lengths.

One of the goals of restricting the export of cryptographic products is to limit the availability of cryptography to (hostile) foreign nations. It is, however, questionable whether export restrictions serve any purpose in preventing these foreign nations from acquiring knowledge about strong cryptography. In addition, export restrictions, at least as enforced by the United States, have spawned much research activity within the areas of security and cryptography in other countries.

With the growing awareness of global networking, export restrictions are also quite ineffective. Most export restricted cryptographic software developed in the United States can also be found on many anonymous *ftp* sites in Europe. Again, it is the process of export and import which is limited; other users (not living in the United States) can legally obtain the software from these non-U.S. sites (provided their own country allows personal use of strong cryptography).

2.4. Summary

In this chapter the field of cryptography and the use of cryptographic methods was surveyed. The survey included a description of secret-key and public-key cryptosystems and hash functions. In addition, a large part of the chapter discussed some of the more important legal aspects of applying cryptographic methods in civilian applications.

In all foreseeable future designers will be faced with restrictions on how cryptographic techniques will be allowed used in civilian applications. Designers may have to build wiretapping functions into their systems in such way that law enforcement agencies can perform wiretapping.

—3—

Computer and Network Security

This chapter is concerned with the fundamentals of computer and network security. The focus is on security models and design methodologies underlying most secure computer systems. The chapter introduces the basics of computer security and defines the terms that are used in the remainder of this dissertation. It also serves the purpose of providing an understanding of computer security in general.

A computer system may be vulnerable to certain attacks and thereby make some threats to the system possible. Security services implemented by components of the computer system serve the purpose of preventing these threats from occuring.

Generally, security models define overall and general policies that are used to describe the circumstances when access to resources in a computer system should be granted. Protection of resources and access control policies are implemented using security mechanisms.

Computer security can be discussed in the context of *objects*, *subjects* and the relations between them. An object denotes a resource in the system. It is typically a machine, file, or any other resource that may be accessed. A *subject* denotes an entity that requests access to a resource. The term *principal* is often used to denote the entity that is held accountable for accessing resources, e.g., a computer system, a user, or an organisational unit.

This chapter is structured as follows. First, the concept of threats to computer systems is introduced and the most common security services are described. Following the introduction is a presentation of the most commonly used security models and implementation techniques for secure systems design. The remainder of the chapter describes security in computer networks and techniques for specification and analysis of authentication and key distribution protocols.

3.1. Threats to computer systems

A computer system may be *vulnerable* to certain *attacks*. These vulnerabilities and attacks may make some *threats* to the computer system possible. A threat to a computer system can, thus, be looked upon as something that can have an undesired effect on the operation of the system, and can be caused by either unintentional or malicious events.

A vulnerability is a weakness in a system that makes a threat to the system possible to occur. Identification and removal of vulnerabilities is essential in order to limit the number of threats to a computer system.

An attack is an explicit hostile action by a malicious intruder. It typically involves the exploitation of a vulnerability in a system, and thereby makes a threat occur.

In this chapter, the semantics of the terms introduced above closely resemble the definitions presented in [Amoroso, 1994].

Central to the security of any system is the assumed threat model. Threats to the security of a system can be divided into disclosure, integrity and denial-of-service threats:

Disclosure: The disclosure threat involves unauthorised access to information stored by a computer system, or to information in transit between computer systems. If information is leaked either unintentionally or maliciously to unauthorised users of the system, it is a break of disclosure. Disclosure threats can, for example, range from the release of classified government documents to the movement of banking information between bank loan managers.

Disclosure threats have been researched extensively. In particular, much research has been conducted on how to prevent unauthorised disclosure of information in larger organisations and military agencies. Disclosure threats concerns system security, and only to a smaller extent computer-aided financial transactions.

Integrity: The integrity threat involves that information is subject to unauthorised changes. Unauthorised changes to information can be either unintentional, or caused by a malicious intruder. Modification can, for example, range from the modification of sensitive government documents to the sensitivity of the correctness of patient medicinal dossiers in a hospital.

Another example is the integrity of computer programs. Unauthorised changes to programs can have major consequences for the protection of applications carrying out, e.g., financial transactions.

Denial of service: Denial of service threats are among the least understood threats to a computer system [Needham, 1994]. The denial of service threat involves that authorised access to some components of a computer system, or some information stored by a computer system is blocked by a malicious intruder.

The denial of service threat can be illustrated by the communication in transaction systems. In a transaction system parties may require protection not only against outside forces, but also from each other. If one party, for example, refuses to serve one particular participant, it may have e.g., economical consequences for this participant.

3.2. Security services

The level of security a system can provide is defined and measured by estimating how well this system can handle specific threats. Security services implemented by the system make computer systems less vulnerable to attacks.

A secure system can offer one or more of the the following services: authentication, confidentiality, integrity, availability and accountability [Canadian System Security Centre, 1993]:

Authentication: An authentication service is concerned with establishing the true identity of a subject requesting access to an object, or to verify whether a claimed identity is authentic. The reply to an authentication request is the proper name of a principal (or sometimes a proxy acting on behalf of the principal). It can be the name of a user, or typically, the principal held responsible for the identity in question. In authentication services lies also the responsibility of detecting principals masquerading as other principals.

Authentication is a fundamental security service. Unless requests to access objects can be properly authenticated, other security services are difficult to provide.

Confidentiality: A confidentiality service is concerned with the protection of information against unauthorised access. It can be to protect information stored by a computer system against disclosure, or protection of information transferred between computer systems. Meta-information, such as information about traffic flow in a system, may also be subject to confidentiality. The confidentiality service makes computer systems less vulnerable to disclosure threats.

Integrity: An integrity service is concerned with protection against unauthorised modification. The integrity service is concerned with information that is already be public, but where unauthorised changes to the information is considered a threat. Integrity services protect against integrity threats.

Availability: The availability service is concerned with accessibility of objects in a computer system, and protection against both denial of service attacks and general failures in the system. Availability is an area of study which has much in common with research on fault-tolerant and dependable computing.

Accountability: Accountability services are concerned with monitoring and auditing of authorisations, and serve in creating logs of access requests and services granted. Accountability concerns can range from ensuring that only authorised users are given access a resource to general monitoring of system activity.

Accountability is closely related to *nonrepudiation*: preventing the successful denial by a subject that some operation was carried out.

Accountability services are important also after an attack has occured. The auditing and accounting logs are typically the only sources of information left (if any) describing the incident in question.

3.3. Security models

Security models are used to define policies and the relationships between subjects and objects in a computer system. In particular, security models can be used to describe under what conditions a given subject may have access to a particular object.

An early description and comparison of formal models for computer security can be found in [Landwehr, 1981]. A more up to date description and comparison of security models can be found in [Amoroso, 1994]. The more influential work on security models are security labels (see, e.g., [Department of Defense, 1985]), the Bell-LaPadula Disclosure Model [Bell and LaPadula, 1973], and the Clark-Wilson Integrity Model [Clark and Wilson, 1987]. In the following, these models are described in more detail.

Labelling is a method of classifying access to information based on security levels and categories. A security level is a hierarchical attribute associated with objects in a computer system. Security labels denote the degree of sensitivity of the object in question. Organisations may define different hierarchical structures for security labels. A well-known example is the military grading hierarchy which defines the following security levels: unclassified, confidential, secret, and top-secret.

The subjects and objects of a system can also be divided in a non-hierarchical fashion by grouping the entities into *categories*. The subjects and objects are grouped into categories according to their degree of sensitivity. In an organisation, for example, each department may constitute one category, and each employee working in a particular department as belonging to a category.

Security labels are attributes associated with each object in a computer system. A security label of an object may consist of a hierarchical security level, and zero or more non-hierarchical categories.

The Bell-LaPadula Disclosure Model (BLP) is one of the most influential security models published. The BLP model is based on an ordered set of security levels, subjects and objects at different levels, and a dominance relation between them.

In BLP, the dominance relation is used to validate classification levels in access requests. Since the security levels are ordered, the dominance relation determines whether a subject at one security level is authorised to access an object at a different security level.

Central to the BLP model are two rules; one for read-requests and one for write-requests. The *no read up* (NRU)[1] rule states that a subject can only read information in objects at lower security levels. If the security label of the subject dominates the security label of the object in question, read access to the object should be granted to this subject. The *no write down* (NWD)[2] rule states that a subject only can write to objects at higher security levels. If the security label of

[1]The NRU rule is often referred to as the simple security property of the Bell-LaPadula Disclosure Model.

[2]The NWD rule is often referred to as the $*$-property of the Bell-LaPadula Disclosure Model.

the subject is dominated by the security label of the object in question, write access should be granted to this subject.

The NRU and NWD rules of the BLP model require that certain criteria are met in an implementation. In particular, the *strong tranquility* property of BLP requires that the security labels of subjects and objects in a system are not changed during system operation. Since this property would render a system relatively inflexible, a *weak tranquility* property only requires that security labels are not changed in any way that may violate any security policies defined in the system.

In BLP, another concern is atomicity. Subjects and objects should not have their labels changed during access requests, e.g., while reading from or writing information to objects at different security levels.

The Clark-Wilson Model Integrity Model (CW) is the result of a study on how commercial organisations control the integrity of paper-based resources in their normal business practice. The research resulted in a model for integrity of data items in computer systems.

In the CW model, the set D consisting of of all objects in a system, is partitioned into two disjoint sets consisting of *constrained data items* (CDI) and *unconstrained data items* (UDI). Objects in UDI do not have any integrity, while objects in CDI have integrity. UDI data items can be upgraded to CDI data items. No object in the system can be a member of both the CDI and UDI sets.

Subjects may request *transformation procedures* (TP) on data items. A transformation procedure consists of one or more atomic actions. Each atomic action is a state transition that may result in updates to some data items. One such transformation can be to upgrade a UDI data item to a CDI data item. An integrity validation procedure that validates whether a CDI has integrity must be available in the system. A TP can only be invoked by an (authenticated and) authorised subject.

With the rules of the CW integrity model, the integrity validation can be controlled. The rules govern how, and by whom a CDI can be changed and UDIs upgraded to CDIs. Since the the CW model is extracted from business practice, auditing is also part of the model. One of the rules states that each TP application must cause information sufficient to reconstruct the application to be written to a special append-only CDI.

3.4. Protection and access control mechanisms

Most secure systems are based on the access control model [Lampson, 1971]. Central to the access control model is the access matrix. Figure 3.1 illustrates a simple access control matrix M containing access rights for the subjects $S_1 \ldots S_n$ to the objects $O_1 \ldots O_m$. In this model, the access rights of subject S_i to object O_j is determined by the contents of the access matrix element M_{ij}.

The access matrix of a system is usually sparse, and many of the elements are empty. The sparseness of access matrices has led to alternative methods of implementation. An access matrix can be viewed either by its rows or its columns,

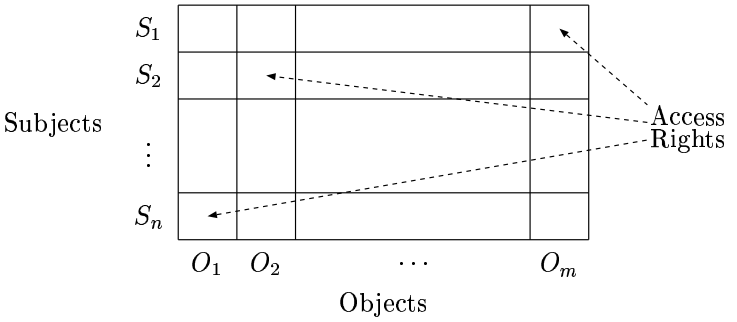

FIGURE 3.1. Illustration of the access matrix and the relation between subjects, objects and access rights.

and this leads to implementation strategies using either capabilities and access control lists respectively.

The reference monitor is a conceptual model of how access to a specific object should granted. A reference monitor can be considered a filter that either allows a request from a subject to access an object, or denies access.

Each (protected) object is *encapsulated* by the reference monitor. The method by which an object is encapsulated may vary. For example, it can be achieved by applying a cryptographic transformation to it. It can also be achieved using a physically protected or tamper-proof machine. Encapsulation also dictates modularity, message passing, and access points as in an object-based client-server model.

An Access Control List (ACL) is an expression of a policy for access to a specific object, and are stored with objects in the system. For a given object, it names the subjects that can access it, and the *access rights* that each subject has to the object in question. The subjects on an ACL can be simple or compound. Simple subjects refer to either users or a service. Compound subjects refer to groups or rôles (see, e.g., [Abadi et al., 1992]).

A typical example of a simple access control list implementation is the Unix ACL scheme and its file permission bits:

```
drwxr-xr-x 4 arne huygens 1024 Aug 27 15:39 /usr/users/arne
```

The file */usr/users/arne* (in this case a directory file) is owned by user *arne* and belongs to group *huygens*. The owner has *Read-Write-Execute* permissions, members of the same group have *Read-Execute* permissions, and all other users who have the same access rights. In Unix, a distinction is made between Read and Execute permission: a subject may request the Unix operating system to execute the contents of a file, but the same subject may not have the necessary rights to read the contents of the same file. Read-permission, however, may allow a subject to copy the contents of a file to this subject's own protection domain where it can modify the ACL of the copy and add the necessary execute rights.

A capability is an object identifier and associated access rights to an object protected by a computer system. Access to the resource in question is granted when a subject can present a valid capability for that object. Capabilities are, in contrast to access control lists, typically stored with the subjects in the system.

A capability can also itself be considered an object that the computer system must protect against unauthorised access. Disclosure of a capability to unauthorised parties is an threat in capability based systems. Integrity threats are also a concern, and the contents of a capability would have to be protected against tampering.

Capabilities are typically implemented using large integers that are generated from the object identifier, access rights, and random numbers to make them harder to guess, or by using cryptographic methods. In centralized systems creation, propagation, and revocation of capabilities are often controlled by the operating system. In distributed systems, cryptographic protocols are mainly used to achieve this.

3.5. Secure kernels

Secure kernels are frequently used to implement secure systems. The security kernel is a component of a system that provides security services to the rest of the system. In modern computer systems, a security kernel is either the entire monolithic operating system kernel of that system, or a subcomponent of the operating system. Important goals are to provide a consistent interface to applications (clients of the operating system), and to minimize the trusted computing base (TCB) of the system. Below follows a more elaborate description of trusted computing bases.

A TCB is the collection of software, firmware, and hardware involved in enforcing the security policy of a system. The TCB can be structured hierarchically, and its components may not be an isolated part of the system. Parts of the TCB can be built in a distributed fashion and located at different places of a system. The key issue is that the integrity of a system depends on all components of the TCB.

A goal is to minimize the TCB of a system—the reason being that it is far easier to build assurance in a small TCB than in large TCBs that consist of more components of a system.

A TCB is usually implemented as a security kernel, and this approach is depicted in Figure 3.2. In the example, the security kernel is partially implemented in hardware and partially software. Only some of the hardware and software of the operating system is part of the TCB. In the figure, applications do not contribute to the TCB of the system.

The reference monitor is an abstract machine, and a subset of the TCB, that mediates accesses to objects by users and processes. The purpose of a reference monitor is to ensure that the information flows between users, processes, and objects are mediated and valid. For any protected object to be manipulated, the reference monitor must be invoked.

A variant of the TCB is the Controlled Application Set paradigm for trusted system [Sterne and Benson, 1995]. The CAS paradigm is based on the premise that every software component that can manipulate sensitive information is potentially

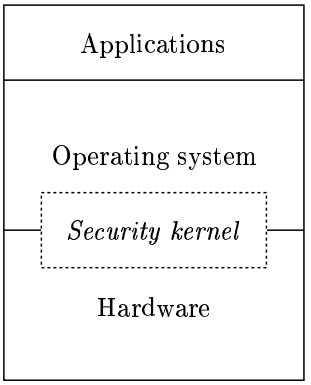

FIGURE 3.2. Security kernel approach

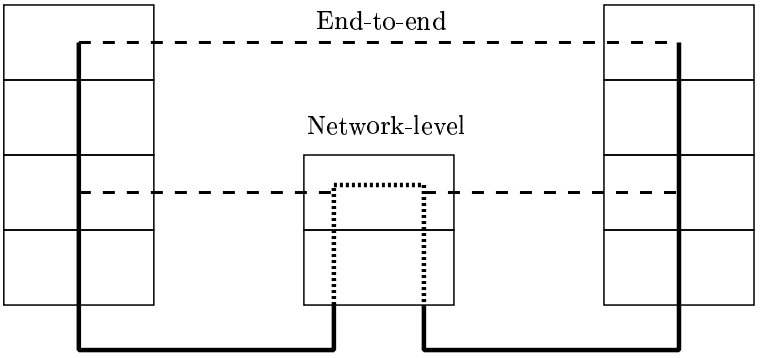

FIGURE 3.3. Network-level and end-to-end protocols

security relevant and must be controlled and protected by automated mechanisms. This counts for applications that have no special access control privileges too.

3.6. Security in computer networks

In layered network protocol designs, each layer provides service abstractions that higher-level layers can access through well-defined service access points (SAP). Each layer requests services provided by lower layers of the protocol. The number of layers, the purpose of each layer, the service abstractions each layer should provide, and that each layer requires from lower layers, has been subject to much research activity and standardisation efforts.

An overview of computer network protocols can be found in [Tanenbaum, 1988], and [Rose, 1990] contains a thorough description of the ISO Reference Model for Open Systems Interconnection (OSI).

High-level network protocols can be divided into network-level and end-to-end protocols. This division is illustrated in Figure 3.3. End systems are nodes in the network containing both end-to-end data communication layer services and application layer services. More generally, in [Rose, 1990, pages 39–40], it is being stated that end systems are where the applications live.

The transport layer (TP4) in the OSI Reference Model is considered a true source-to-destination end-to-end layer [Tanenbaum, 1988]. The Transport Control Protocol (TCP) [Postel, 1981b] of the TCP/IP protocol suite is also considered an end-to-end layer. An analysis and discussion of security related problems in high-level network protocols can be found in [Voydock and Kent, 1983].

Security in end-to-end layers is concerned with security services for end systems. In particular, authentication of end systems, and integrity and confidentiality of data communication between end systems. End-to-end security may for some higher-level application services reach into the application domain. GSSAPI [Linn, 1993a], for example, is a generic security service application program interface for end-to-end security protocols.

Network-level protocols are used for data communication between end systems and intermediate systems in the network. Nodes in the network that provide network-level services may not have higher-level end-to-end level protocols resident.

Security concerns in network-level protocols are typically associated with authentication of the source and destination addresses of a datagram, and integrity of datagram contents. Confidentiality may be an issue, but may also be deferred to higher-level protocols.

Two essential functions of high-level networks are secure network communication between hosts and protection against attacks by a third party. One threat model for computer networks commonly referred to is one in which two mutually trusted end systems communicate over an insecure medium (see, e.g., [Voydock and Kent, 1983]). Attacks aimed at either one or both of the end-points may be launched by a third party that has access to the communication medium.

3.7. Cryptographic protocols

A cryptographic protocol is a specification for the format and relative timing of information or messages exchanged between communicating parties, where cryptographic mechanisms are used to guarantee the meaning of messages [Gong, 1990]. Methods such as cryptography can be used to preserve the integrity, secrecy, origin, destination, the order and the timeliness of messages exchanged between communicating parties.

Cryptographic protocols are essential to security in distributed systems where parties communicate by exchanging messages. In such environments, the meaning of a message can only be derived from the message itself, and cryptographic mechanisms and algorithms can be used to protect the message contents and preserve the causality of messages that are exchanged during a protocol execution.

The basic components used to develop cryptographic protocols are messages, nonces, encryption keys, and other administrative information. The format of each

message should be unambiguously defined. Nonces are freshness identifiers used to establish timeliness in a protocol execution. The execution of a cryptographic protocol consists of a finite sequence of messages exchanged between the participants in the protocol.

Due to subtle errors in their specification, cryptographic protocols often fail to meet the intended goals. The reasons are typically that either the assumptions about the initial conditions are wrong, or that information that should have remained confidential is revealed during the execution of the protocol.

3.7.1. Key management. When sharing of (cryptographically protected) information is a requirement, key management is an important aspect. If an object is encrypted with a particular key, the subjects with knowledge of that key are able to decrypt it. In networked environments, where principals are not physically located at the same place, key management becomes an issue. If two parties cannot meet to exchange the necessary keys, these keys must be exchanged encrypted by other keys that already are shared between these parties, or be distributed via a third party trusted to handle keys properly.

Two common ways to provide key distribution is via on-line key distribution centers (KDC), or by using trusted certification authorities (CA). In the following these two approaches are discussed in more detail.

Key distribution centers (KDC) are used to provide the keys needed by other principals, and can provide keys in an on-line fashion. It is often an implementation of a trusted third party (TTP) approach to key distribution where all involved parties would have to trust the KDC to provide proper keys for all further communication.

The use of a single key distribution center is an approach that does not scale to a large network of computers, or network topologies that span multiple organisational domains. A single key distribution center can easily become the bottleneck in a large system. In addition, organisations might want to enforce different security policies.

The use of a certification hierarchy has been proposed as a solution to the scaling problem of single key distribution centers. The idea is to divide responsibility between different organisational units and let each group manage a subset of all users and services.

An essential component of a certification hierarchy is the Certification Authority (CA). A CA is an organisation (or subdivision of an organization) responsible for verifying the security attributes of computer system users, and entering this verified information into the computer system [Roe, 1992].

Typically, there will exist more than one CA. These will be structured into a hierarchy according to their trust relationships. In such an organisation there will exists different kinds of CAs, and these can be divided into the following three groups:

- Top-Level CAs establishing cross-certification between countries.
- Policy Certifying CAs certifying organisations at the national level.

- Organisational CAs certifying organisational units and individuals belonging to organisations.

The structure of cross-certification at the top-level of a certification hierarchy can be rather complex and express the trust relationships between certification authorities in different countries and trust relationship between them.

In order for a CA to be trustworthy, it must be assured that the operational procedures comply with prescribed obligations of certification authorities as put forward by laws and security policies. These obligations can be divided into the following three categories:

- A CA shall only make statements it believes. That is, a CA has an obligation to not make false statements.
- A CA shall only make statements based on justified beliefs.
- A CA shall not make ambiguous statements. The language used to express statements shall assign only one meaning to each statement made.

Another essential component for the operation of certification authorities is the certificate. A *certificate* is a signed statement issued by a certification authority. It is typically used to bind names to public keys. The most common certificate format is the identity-certificate defined by the X.509 standard [CCITT, 1991].

A CA is granted jurisdiction over a subdomain of the global name space and creates identity-certificates for organisational units and users in that subdomain.

Each user of the system will trust different CAs to different extents. In particular, trust in other certification authorities is established indirectly through the certification hierarchy on the assumption that each certification authority complies with the obligations mentioned above.

A different approach to the use of certification authorities is the one employed by PGP [Zimmerman, 1994] (see also Section 3.8). When using PGP, a user is his own certification authority and free to certify keys belonging to any other user. The authentication space of PGP is flat, and the notion of trusted third parties does not exists.

The reasoning behind PGP-like authentication spaces is to base authentication of keys on a *web of trust*. If a key is signed by another user this user trusts, he may choose to trust the authenticity of this key too. The lack of structure as in X.509 means that it is often not possible to determine the trust in a particular key belonging to an unrelated user. This is also part of PGP's design; trust should be built on personal/social relationships, and not on a third party that must be trusted by default.

The SDSI [Rivest and Lampson, 1996] approach to hierarchic certification infrastructure (see also Chapter 6) borrows much from X.509. However, it also incorporates features of PGP-like certification of keys. In particular, also in SDSI each user is his own certification authority. In addition, SDSI users can build their own certification hierarchy, and introduce global roots (e.g., X.509) via references to names in a local name space. Since principals in SDSI are keys acting on behalf of users, there is no clear distinction between authorities (as in X.509) and individual users (as in PGP).

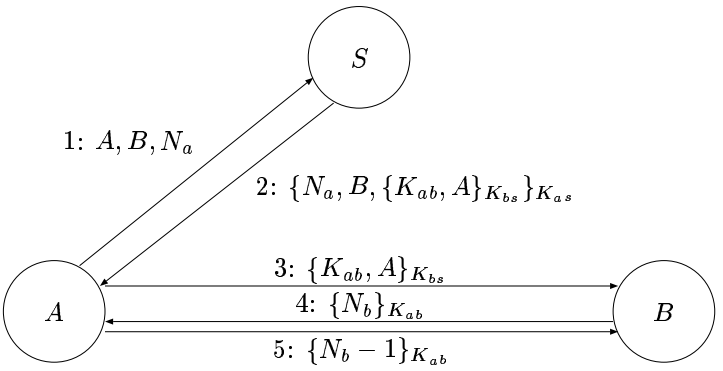

FIGURE 3.4. The Needham-Schroeder protocol

3.7.2. Authentication protocols. A class of cryptographic protocols are concerned with authentication of principals, and optionally, distribution of cryptographic keys that can be used for secure communication between the authenticated parties. A survey of authentication protocols can be found in [Liebl, 1993].

Authentication is the process of establishing the identity of the sender of a message. The ability to establish the identity of principals is essential and fundamental to the correct operation of any access control scheme.

Authentication protocols are often described as formulas or mathematical expressions. In this dissertation, the notation found in [Burrows et al., 1991] is adopted.

One of the first authentication protocols published is the Needham-Schroeder protocol [Needham and Schroeder, 1978]. The protocol authenticates a client to a server using a third party on-line authentication and key-distribution server. Both the client and the server share secrets with the authentication server prior to any communication between them. The authentication server chooses a session key for the communication between the client and the server and presents the key the client as a ticket. The client uses this ticket to authenticate itself to the server.

Figure 3.4 illustrates the Needham-Schroeder protocol. In Message 1 the client A requests a ticket for service B from the authentication server S. Message 2 is sent from S to A and contains the ticket $\{K_{ab}, A\}_{K_{bs}}$ with a session key for subsequent communication between A and B. The message is encrypted with the key K_{as} which is shared between A and S. A nonce N_a which is initially sent to S in Message 1 is also embedded in the reply to A in Message 2 and is used by A to verify that the reply from S is part of the current protocol run. Message 3 is the initial authentication request sent from A to B and contains the ticket received from S. Since the ticket is encrypted with the key K_{bs} which is shared between B and S, it can be decrypted by B to reveal the new session key K_{ab}. Message 4 and Message 5 is a challenge-response for B to determine that A is present, and that Message 3 is not a replay of an old message. The nonce N_b is encrypted with

the new session key and sent to A. The response from A back to B again is the $N_b - 1$ also encrypted with the session key and is used to convince B that A is participating in the same protocol run, and that Message 3 is not a replay.

Since its publication, the Needham-Schroeder protocols has been studied extensively, and many other protocols have been influenced by its design. The original protocol, however, contains a weakness; upon reception of Message 3, B cannot tell whether the encrypted session key received is fresh, or whether A is sending an old session key [Denning and Sacco, 1981]. Furthermore, if A's private key is compromised it can be used to obtain tickets to many services and use them even after A's key has been changed [Bauer et al., 1983].

3.7.3. Protocol design and analysis. Authentication protocols typically consist of the exchange of just a few messages, and their specifications can in this respect be compared to small computer programs. Despite this apparent simplicity, the number of flawed protocols described in the literature shows that it is easy to get such protocols wrong (see [Burrows et al., 1991]).

An overview of state of applying formal methods to cryptographic protocols can be found in [Meadows, 1995]. In [Gong, 1995], lower bounds and optimal implementation of network authentication protocols are discussed. Formal requirements for key distribution protocols are discussed in [Syverson and Meadows, 1995].

Formal analysis techniques have been developed to encounter errors in the specification of authentication protocols and to obtain proofs of their correctness. Although a proof of correctness can be obtained, this does not necessarily mean that a proof of security is also obtained.

Another approach to protocol design is to rely on knowledge of experts in the field and thereby (hopefully) avoiding some of the big blunders of other designers. Expert knowledge can be presented to developers as design rules for prudent protocol design.

In order to provide system designers and researchers with tools to specify and analyse authentication and key distributions protocols, a variety of formal specification languages and verifications tools have been developed. Different approaches have been proposed, ranging from the use for formal languages for secure protocol specification to the use of expert systems and protocol analysers to determine the outcome of a protocol execution.

One of the first authentication logics proposed, and one of the most influential ones is the the BAN logic of authentication [Burrows et al., 1991]. The BAN logic is put forward as a propositional logic and it is used to reason about the beliefs of principals in the course of authentication.

When the BAN logic is applied to a protocol, all messages of the protocol are rewritten as expressions of the logic, and the initial assumptions about beliefs are stated as expressions of the logic. With these expressions and BAN's postulates, the state of knowledge of each principal involved in a protocol execution can be derived.

The BAN logic analysis of a protocol proceeds in steps. The first step is to reformulate and idealize the protocol to be analysed as expressions of the logic.

This step is typically carried out manually by the protocol designer and it is not always straightforward. The second step is to state the initial assumptions underlying the protocol. Also, it is the protocol designer's knowledge and understanding of the assumptions underlying the protocol in question that are transformed into expressions of the logic. The third step is to use the postulates of the logic and the protocol expressions (the idealized protocol messages) to reason from assumptions to conclusions, and the state of knowledge and beliefs of entities in the system after a protocol execution has taken place.

The BAN logic has been widely criticized. The main criticism is concerned with the secrecy assumption underlying the use of BAN. The BAN logic assumes that secrets (e.g., keys) remain secret during the execution of the protocol being analysed. Since the secrecy of a secret may depend on whether the protocol in question is secure, the secrecy assumptions cannot be used to derive the security of the protocol unless a separate mechanism can justify this assumption (see, e.g., the discussion in [Gong and Syverson, 1995]).

Several extensions to the original BAN logic have been proposed. In [Abadi et al., 1990], an extended version of BAN that can also be used to reason about channel security is described and how the extended logic can be used to reason about authentication and delegation with smart cards.

A more extensive authentication logic than BAN is the GNY logic of authentication (proposed in [Gong et al., 1990]). The main difference between BAN and GNY is that the latter distinguishes between possessions of formulas and beliefs. In this way it is possible to distinguish between the contents of a message and the information implied by it.

In [Gong and Syverson, 1995], a class of secure protocols called *fail-stop* protocols is described. This approach can be considered as an example of the application of both formal methods and structured design rules in protocol design. The novelty of protocols developed using the fail-stop approach is that they are resistant against active attacks. That is, any such attempt will cause an early termination of a protocol run. Therefore, when analysing fail-stop protocols, it is only necessary to consider the effects of passive attacks.

A protocol is fail-stop if any attack interfering with a message sent in one step of the protocol will cause all causally-after messages in the next step or later not to be sent. The messages of the protocol are organized into a directed acyclic graph where each message represents an arc, and each directed path represents a sequence of messages.

A protocol should only be considered fail-stop if the content of each message has a header containing the identity of its sender, the identity of its intended recipient, the protocol identifier and its version number, a message sequence number, and a freshness identifier. Each message should also be encrypted under the key shared between its sender and intended recipient. If a public-key system is used, then each message should instead be signed by the sender's private key, and then optionally encrypted under the public key of the recipient. Honest processes should follow the

protocol and ignore all unexpected messages, and halt any protocol run in which an expected message does not arrive within a specified timeout period.

A more generalized class of protocols called *fail-safe* protocols also exists. A fail-safe protocol is one in which the response to any message that make the protocol not fail-stop is safe.

Many protocols that rely on nonces as freshness identifiers for messages are not fail-stop. For example, there is no way that the recipient of the first message a pure nonce based protocol can determine whether the message is fresh unless a trusted third party providing a beacon (for example, in the form of a digital time-stamping service [Haber and Stornetta, 1991]) is used. To proceed with the protocol the recipient would have to respond to the message as if it is fresh, or terminate the protocol to satisfy the fail-stop property. If a response to the message allows the protocol to proceed in a fail-stop manner one could argue that it also is safe to do so. The process of analysing fail-stop protocols is divided into the following steps:

(1) Verify the fail-stop property. In this step it is verified that the protocol to be analysed satisfies the requirements for fail-stop protocols mentioned above.
(2) Validate the secrecy assumption. Since active attacks will halt a fail-stop protocol, validation of the secrecy assumption is concerned with establishing that no other party can obtain any secrets by recording and manipulation of the messages exchanged during a protocol execution (e.g., by applying the possession rules of the GNY logic).
(3) Apply BAN-like logics. The previous step validated whether the protocol satisfies the crucial secrecy assumption required to apply the BAN logic, and BAN can now be used to capture other potential weaknesses inherent in the analysed protocol.

A different approach to secure protocol design is to rely on accumulated knowledge that have been already gained by experts in the research community. The strength of such an approach is that one can avoid the blunders others have already made.

Knowledge about flaws discovered in other authentication and key distribution protocols is valuable when new protocols are being developed. Several design principles to avoid common problems with authentication and key distribution protocols have been proposed in the literature (see, e.g., [Abadi and Needham, 1994]). Some of these principles deal with encryption and timeliness of messages, while others deal with the meaning of the contents of a message.

Generally applicable design principles frequently seen are *robustness* and *explicitness*: when designing secure protocols one should be explicit about all security properties such as naming, message typing, freshness, function of encryption, starting assumptions and what the protocol should achieve. The fail-stop protocol approach presented above is one example of applying knowledge and experience to devise a method for secure protocol design.

In public-key systems, another important principle is concerned with the order of sign and encrypt operations. It is considered prudent to sign a message before

encrypting it, since this signifies that the sender knew the message contents before it was encrypted [Abadi and Needham, 1994]. A variety of engineering techniques for public-key systems are described in [Anderson and Needham, 1995b].

3.8. Applications

Currently cryptography and computer security is a hot research topic, and many security systems and applications have been developed. This section describes some of the more influential application systems.

Kerberos [Steiner et al., 1988] is an authentication system for open networks used to authenticate clients to servers using an on-line authentication server. Kerberos was developed as part of project Athena at MIT and it has been widely used for authentication on the Internet. In Kerberos, authentication and key distribution is based on an enhanced version of the Needham-Schroeder protocol.

The initial target environment for Kerberos was a university campus area where users were required to authenticate themselves to remote services available on campus.

The core of the Kerberos system is an on-line authentication and key distribution server. Since the machine running the Kerberos server holds the secrets of all users, it must be physically secured. If this machine is compromised, an attacker can to masquerade as any user or service in the system.

When a user logs in to a machine, an initial request for a Ticket Granting Ticket (TGT) is sent to the Kerberos server. The ticket is encrypted with the user's password and returned to the machine from which the request originated. If the user can encrypt the message with his password, the Kerberos login is successful. Note that this scheme is vulnerable to off-line attacks; an intruder can request the encrypted ticket for any user and launch an off-line brute force attack on the key. In [Lomas et al., 1989], methods for avoiding such attacks are discussed in more detail.

Users who have obtained a TGT can use it to request remote services, e.g., to login securely to a remote machine. Before a service request can take place a ticket for the login service must be obtained from the Kerberos server by the user and presented to the remote machine. Kerberos implements access control lists for each service in the system and users must be added to the access control list of a service before a successful access of that service can take place.

In [Bellovin and Merrit, 1991], some of the limitations of the Kerberos protocol (Version 4) are discussed. The most important part of their criticism is concerned with the use of timestamps. Since Kerberos relies on loosely synchronized clocks, clock synchronization in the system must also be secure. The obvious threat is that if a machine, somehow, can be tricked into believing in the wrong time, Kerberos authentication messages can be replayed against this machine. The more recent Kerberos Version 5 protocol contains improvements for some of these issues.

For more information on the risks of depending on synchronized clocks, see, e.g., [Gong, 1992]. A taxonomy of replay attacks is presented in [Syverson, 1994].

Pretty Good Privacy (PGP) [Zimmerman, 1994] is a security enhanced cryptographic software application available for a wide range of different platforms. PGP is based on public-key cryptography using the RSA [Rivest et al., 1978] algorithm and message digests to provide digital signatures and public key certificates. Fast data encryption is achieved using the IDEA [Lai, 1991] shared key encryption algorithm.

PGP uses a "guerilla approach" to key management. Essentially, each PGP user can be a certification authority and sign the public keys of other users. This creates a web of trust between PGP users. This approach differs significantly from the hierarchically structured certification authorities found in other global authentication systems.

Standardization efforts also takes place within the security communities. The CCITT recommendation X.509 [CCITT, 1991] is the security framework of the X.500 series of recommendations for a global directory service. The directory service itself is a distributed set of servers that maintains a database containing information about users (and services).

The X.500 directory service provides mappings from user names to attributes such as network addresses, and personal information, i.e. public keys. The X.500 service is expected to be widely used in conjunction with other distributed services. Privacy Enhanced Mail (PEM) [Linn, 1993b] is an example of one such service.

The X.509 recommendation defines a security framework for the provision of authentication services by the X.500 service to its users. At the core of this framework is the public-key certificate associated with each user. The X.509 certificate is a piece of information that contains the public-key components of a user together with administrative information that describe the name of the owner and the validity of the certificate. Attached to each certificate is also a digital signature created from the information mentioned above and signed by a certification authority. The certificates can be stored by a X.500 naming service or in some other service that does not need to be trusted.

3.9. Summary

This chapter discussed the fundamentals of computer and network security, including design methodologies for developing authentication and key distribution protocols.

In distributed systems, the use of cryptographic methods and protocols is essential to the provision of security services. The most commonly used threat models were sketched together with an informal description of commonly used security services for networked environments.

—4—

The Trust Architecture

In computer and network security, a commonly used threat model is to assume that end systems are located in secure areas while the network itself is not. Many existing transaction systems have been developed using this threat model and it has proved useful when end systems mutually trust each other. However, the model fails to provide true end-to-end security for electronic transaction processing and commerce in other settings, e.g., Internet-like environments, because the assumptions made about the end systems generally do not hold.

Often the problems are rooted in poor understanding of what constitutes an end system, or in application features that violate some of the underlying security assumptions of the system, or simply, in poor maintenance procedures. For example, it is common for ordinary users to install new software on their personal computers, or have such software automatically downloaded to their machines via computer networks. Since it may not always be possible for users to determine whether new software is harmless, or whether it contains malicious code created by an intruder, there is a risk that end systems are compromised in subtle ways.

Compromised end systems and malicious users are concerns for electronic-commerce applications that handle information representing real values. If an end system that is used for electronic-transaction purposes is compromised, a malicious intruder might be in a position to intervene in executing transactions, or even worse, conduct new transactions on the user's behalf without this user's consent. It is also a risk that fraudulent services accessed via public networks may try to trick "victims" into signing agreements on the wrong premises.

Ideally, an electronic transaction system should provide true end-to-end coordination of transactions between the user and a computer without the possibility of malicious intervention by a third party. An end system residing in a secure area which is under direct control of its user should be used to engage in transactions with remote services. Only when the end system is under direct control by the user and this user decides to engage in transactions, critical operations such as signing authorisation statements should be activated.

This chapter outlines the architecture of an end system for electronic transaction processing that protects its users under the threat model described above. The architecture is called Trust, and describes a computer system that provides a trusted path between a user and his signing keys, and incorporates security functions into the human-visible computer interface. The architecture supports interaction between the user and a device that enables a transaction to be conducted securely

between that device and another system. The transaction is not conducted by the device unless it is explicitly authorized by the user to do so.

In the remainder of the chapter, the Trust architecture is described in more detail. First, the motivation and underlying design principles are elaborated on. Then the overall architecture is sketched and details of each architectural component is discussed in more detail. Included in the discussion is also a description of the operational mode of the design. The chapter ends with a discussion on how the design can be incorporated into existing computer systems.

4.1. Motivation

Traditionally, protocols and mechanisms for secure communication in open networked environments have been incorporated into either the network layer (network-level security), or higher level applications protocols (end-to-end security) [Voydock and Kent, 1983]. In both approaches, the threat model assumes a hostile third party located in the network (see also Chapter 3).

Network-level security services can achieve secure communication between pairs of mutually trusted hosts over insecure networks. However, it provides typically no application-specific security assurances except, perhaps, for mutual authentication.

Application-level security services improve the situation. When security abstractions are no longer transparent to the applications, a wider spectrum of flexibility is obtained. However, since most applications executing on personal computers cannot be considered secure by themselves, security still depends on protection mechanisms implemented by the hardware and operating system.

Dynamically loaded programs and agent technologies require that the (operating) system can distinguish between trusted and untrusted application components. In general, it is difficult to determine whether a program is malicious when it is actually being executed. Therefore, neither the operating system nor the applications can be trusted to protect keys belonging to users.

Techniques that are often used to construct trusted systems are the security-kernel approach and the use of trusted computing bases (TCB). In these approaches, a subset of the system provides security functions to less trusted components of the system. System security and integrity critically depends on the correct functioning of the TCB.

Users must anyhow be able to mediate authorisation requests to the signing functions residing in the TCB, and only enable them to sign authorisation statements when the user requests it. Unless there is a *trusted path* from the human user to the signing functions (and keys) that only the user can access, there is a risk that operating system components or applications can circumvent the security functions and compromise the user.

Poor operating system security could even be in the interest of the owner of a system if it can help him to get out of a contract by claiming that his keys have been compromised without his knowledge. It is equally important to guarantee that users cannot successfully deny their own signatures.

A necessary design methodology is to build an end system with a secure mode of operation that can only be activated via external means and by explicit actions by the human user, e.g., a button located on the keyboard that must be pressed before a transaction can be authorised. The central idea is to design a trusted path between the human user and the signing functions that is not under control of the operating system or any of the applications. This approach makes the use of security functions more explicit, and brings it under control of the human user. If carefully designed, the authorisation functions do not need to be controlled by the operating system.

A system with a secure mode of operation lets applications executing on the user's computer make use of the computer's keyboard and screen to render transaction related information and to request authorisation statements from the user. The two issues of concern are the presentation of information on secure channels, and how to guarantee that authorisations are not given unintentionally. The failure semantics should be fail-safe [Leveson, 1995]; the system should not be able to authorise any transactions on behalf of a user unless this is acknowledged explicitly by the user.

The fundamental goal is to devise a secure end-user platform for electronic transactions that protects the integrity of the human-visible user interface and limits access to the signing functions and keys. There are two important issues involved:

(1) Ensure that human users base their authorisation decisions on authentic information, and
(2) protect the signing keys used for authorisation purposes against unauthorised disclosure and usage.

In the following, these issues are discussed in more detail.

4.1.1. The human user. In human-computer interaction, users respond to the information presented to them via human-visible computer interfaces (see also Section 1.3). For a correct decision to be made, the information presented to the user must be correct and unambiguous. If the information presented is incomplete, erroneous or contains ambiguities, the user may make wrong decisions.

Incomplete specification of human-computer interfaces is a common source of failures in safety critical systems [Leveson, 1995]. In a security context, however, the information presented to the user must also be authentic. If, for example, authenticity of the information displayed to a user during a contract[1] signing protocol cannot be guaranteed, the user might sign the contract on the wrong premises. It is not sufficient to guarantee the completeness of a user interface, it is also necessary to protect the user-interface against tampering by malicious users and applications.

Integrity and atomicity are essential properties required to achieve secure user interfaces:

[1] A *contract*, in the terminology adopted here, is a statement signed by the parties it binds.

Integrity: Information presented to a human user should be authentic and not subject to unauthorised modifications. Security mechanisms are required to protect against tampering and to verify the authenticity of the displayed information. Security mechanisms are also required to alert the user about a compromised user interface.

Atomicity: Operations initiated by the user as the response to some displayed information, should be bound to that information, and to that information only. Since the system must enforce that operations are actually executed on the information presented to the human user and not something else, this atomicity property is required.

It is, thus, required that the system is capable of providing a mode of operation in which it guarantees that a particular operation is performed on information presented to the human user and on that information only.

Naturally, a secure user interface must also exhibit other properties that such interfaces should have in order not to confuse the user. In [Jaffe, 1988], three questions are identified as important to the requirement specification for information displayed to a human user:

(1) What event should cause a particular item to be displayed?
(2) Can and should the display of a particular item ever be updated once it has been displayed? If so, what events can cause an update? Events that trigger updates can be:
    - external observables,
    - the passage of time,
    - actions taken by the viewing users,
    - actions taken by other users (in multi-user systems).
(3) What events should cause a particular item to disappear?

4.1.2. Confidentiality of keys. Protecting the confidentiality of digital signing keys is essential for nonrepudiation purposes. If any such key is compromised, statements can be signed without the consent of the proper key owner.

Since cryptographic keys are usually too large to be easily remembered by humans, they are typically stored and protected on secure and tamper-resistant hardware devices.

Smart cards are commonly used nowadays to store secret keys. Most smart cards on the market today have to be activated by a human providing a Personal Identification Number (PIN). Since smart cards do not provide any user interface, the PIN usually has to be entered on a keyboard of a smart card reader.

The use of external input and output devices introduces the risk that the PIN belonging to the user can be intercepted a (malicious) smart card reader. Since the smart card and the PIN together is all that is required to make the card authorise something, there is nothing that prevents a malicious card reader from obtaining multiple authorisations without alarming the card owner.

In addition, the amount (of money) the card is requested to sign for is often shown on the display of the card reader. A card reader operated by a malicious user can display an amount that is different from the one presented to the smart

card for signing. Once the PIN has been obtained from the user, the card reader holds all the credentials to control the card.

When a smart card has a PIN pad and a display of its own, a higher level of security would be achieved. PINs cannot be intercepted any longer because they are only entered on the smart card itself and the card does not divulge them to the outside world. The amount for which the smart card signs is displayed on the smart card's own display. When this information is verified by the user before allowing the smart card to sign, it is more difficult for an attacker to trick that user into allowing the card to sign for the wrong amount. However, such displays are only suitable for a very small number of simple, standard applications.

It is often assumed that smart cards and secure processors are tamper resistant devices. Indeed, manufacturers often claim that their security products are tamper proof. However, in practice it has been shown that most of the so-called tamper resistant security devices available on the market, are not very tamper resistant at all.

A typical example is the cards that are used for pay-TV. Several of these devices have been reverse engineered and their internals published anonymously on public networks. In some cases it turned out that the complexity of the reverse engineering process was rather low.

Designers usually rely on a *work-factor* describing the efforts required to break a given system. Often these estimates turn out to be wrong. In [Anderson and Kuhn, 1996], for example, it is being argued that one should be careful when relying on tamper resistant devices or other so-called "silver bullet" technologies. Various successful attacks on "highly secure" processors available on the market were reported. The costs of carrying out some of these attacks varied from $30 to $100 and a personal computer for the most advanced attack. The time it took to carry out the attack on a "highly secure" device was 3 months including the time needed to survey all necessary literature.

Generally, it is prudent engineering practice to avoid single points of failures, and this is especially important when the likelihood of an successful attack is not known.

4.1.3. Nonrepudiation. When an agreement between two or more parties has been signed by all of them, the parties expect to be able to hold one another to the terms of the agreement. If necessary, this could be by force of law. An important property of a signature on a contract is that the signer cannot afterwards deny successfully that the contract was signed by him. Nonrepudiation is the property that something that happened cannot later be denied. For electronic transactions, it is the information gathered during the transaction and exchanged between the participants of the transaction that will be used to implement nonrepudiation.

Digital signatures (see Chapter 2) are often used to implement nonrepudiation. However, the use of digital signatures for nonrepudiation purposes is based on the assumption that the signing key has not been compromised. Underlying this are assumptions about the keys in use. In particular, it is essential to maintain the confidentiality of the private keys in use. If a private key is compromised, it can be

used by another principal to sign statements on behalf of the key-owner, and thus, defeat the nonrepudiation argument. Similarly, it is also essential to ensure the integrity of the public keys in use. Unless the binding of an identity to a particular public key can be guaranteed, a secure channel to the principal with a given identity cannot be achieved.

4.2. Architectural Overview

In this section, an overview of the Trust architecture is presented. In the Trust conceptual architecture, hardware mechanisms to provide a trusted path between a human user and that user's signing key, and integrity check functions for the visual user interface are incorporated into the architecture. In addition, supporting protocols for contract signing, authentication and authorisation, and network-level security are incorporated into the architecture.

The vital properties to be achieved by the Trust architecture are:

- Electronic transactions can be securely conducted between persons, or between a person and a computer system.
- The parties can negotiate and enter into *arbitrary* contracts.
- A statement or contract is never signed by a device without explicit authorization from its owner.
- Users trust a small device which they own, as well as signed statements made by certification authorities (CAs) and they trust decisions made by arbitrators they choose and trust. No trust needs to be placed in the correctness of any equipment or software owned by others.
- The device the user trusts builds an audit trail that allows the owners to present evidence of contracts to arbitrary third parties, including a court of law.[2]
- In case of a failure during the contract-signing exchange, third-party arbitration is used to resolve the pending transaction in one way or the other.
- As much as possible of the complications of reliably exchanging information over a computer network, obtaining the necessary certificates, and negotiating the contents of the contract, is delegated to the (untrusted) host operating system. The interface of the security device to the user and to the rest of the system is kept as simple as possible.

The Trust architecture follows the Controlled Application Set (CAS) paradigm for trusted systems [Sterne and Benson, 1995] (see also Section 3.5) and distinguishes between controlled application set subjects and untrusted subjects. Underlying the use of the CAS paradigm is the assumption of a TCB that meets requirements for tamper protection, non-bypass ability, trusted path, access to CAS subjects and programs, and functionally correct services.

Access to sensitive information (for example, signing keys) is only granted to some applications that are acting on behalf of CAS subjects and for which some

[2]However, for the evidence to be useful in court it would require legal recognition of digital signatures.

assurance of benign behaviour has been obtained a priori. Similar requirements are also made about the applications controlling the input and output devices of the system.

The security perimeter of the system is constructed in such way that the user (or the operating system) can force the system into a *secure mode* of operation in which functions critical to the processing of electronic transactions can be performed securely. The goal of this approach is to prohibit revealing any sensitive information to applications acting on behalf of untrusted subjects.

4.2.1. Hardware components. The Trust hardware architecture is conceptualized as a *security module*, which consists of a secure processor that is used to support the execution of signature and verification functions. The security module also provides an interface between the security subsystem, and other components of the architecture.

The overall hardware architecture consists of the following components:

Security module: The security module is the core of the human user's trusted computing base. It provides an interface to security functions through which a local computer system can request security services.

Input/output device: The input/output device is the human-visible interface to the computer system, and in particular to the services of the security module.

Smart card: The smart card is an optional device that holds the human user's signing key. The card's functionality is optional as the security module itself may implement secure key storage and the cryptographic functions to sign and verify messages.

Local computer: The local computer system is a standard computer system with CPU, memory, and other resources required to perform computations. It generally constitutes a personal computer. The local computer is wholly outside the TCB of the system.

Remote computer: The remote computer is the peer computer system in electronic transactions, and is typically connected to the local computer system via a computer network. In transactions, the remote computer is typically the merchant's computer.

The architectural components are illustrated in Figure 4.1. The terminal input and output devices and the security module together form the TCB of the system. When the security module is activated, e.g., by the use of a smart card, communication from keyboard to the security module is kept confidential (e.g., only the security module and (optionally) smart card sees the authorisation code), and communication from the output device cannot be modified by other devices on the local computer system.

The main responsibility of the security module is to sign and verify messages during the execution of transaction protocols, and to do this without being subject to interference from the operating system of the local computer system or from

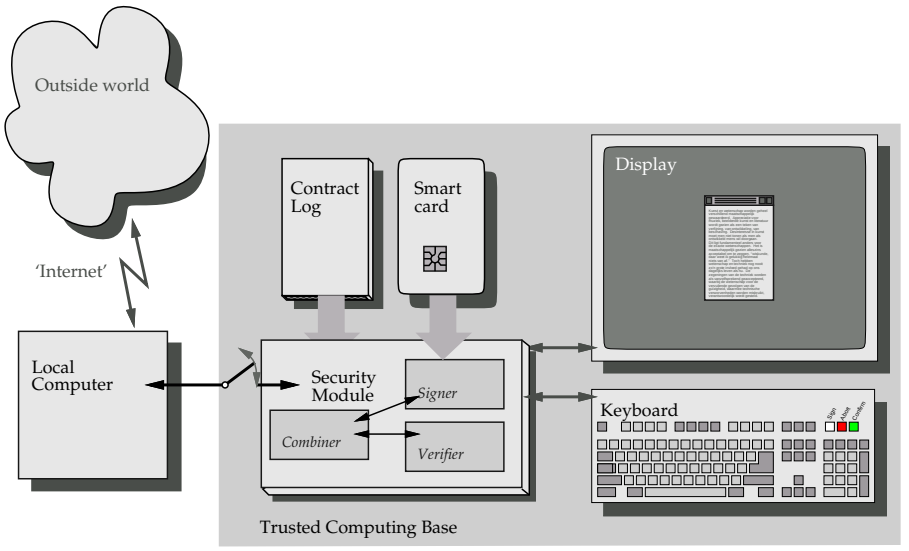

FIGURE 4.1. Architectural overview

applications being executed by the local computer. In this respect, the Trust architecture treats both the operating system and applications as programs acting on behalf of untrusted subjects.

The security module protects user authorisation codes from being accessed by the operating system. Preventing authorisation codes from being accessed by the operating system or applications, helps ensuring that only messages authorised by the user can be signed by the security module.

In order to verify that information shown to the user is correct, the security module provides signature verification functions ensuring that information is rendered correctly on the system's output device.

In order to activate the security module to sign statements, the user must present a correct authorisation code to it. The authorisation code is typically a secret shared between the user and the security module established prior to any system operation.

User authorisation codes that activate the smart card to sign statements require special attention. In particular, the circumstances under which an authorisation code can cause a signature to be created, need to be well understood.

4.2.1.1. *Secure mode.* In the Trust architecture, a *secure mode* of operation is incorporated into the architecture to handle authorisation codes. When entering secure mode, system control is transferred to the security module. There are two important criteria for the operation of Trust's secure mode:

(1) Secure mode can only be activated by user action or a request from transaction software executed by the local computer. A trusted path exists on which the user can signal to the security module that it should enter secure mode of operation. Secure mode of operation, however, can only be terminated through explicit user interaction.
(2) User authorisation codes are presented on trusted paths that are not controlled by the local operating system, and only when secure mode is enabled.

Thus, the main properties of the secure mode of operation are that the user or the local operating system can request it, and transfer control to the security module. However, the local operating system cannot disable the secure mode and return to normal system operation on its own.

Secure-mode activation can be implemented in different ways and generally depends on architecture dependent features. For example, exclusive access to the main system bus by the security module might be required to prevent the operating system of the host system to interfere with its operation.

When the system operates in secure mode, the security module can access resources, e.g., frame buffer memory without being interfered by other devices on the local computer system. Control can later be returned to the operating system for continued normal operation.

4.2.2. Software components. The software components of the Trust architecture are designed to provide transaction protocol handling and logging, authorisation and access control, and network-level security when communicating with remote hosts. All these subcomponents are intended to be executed by the local computer and are all considered untrusted subjects.

In the following an overall description of the software components is presented. A detailed description of each subcomponent is deferred to Chapter 5, Chapter 6 and Chapter 7, respectively.

Transaction protocol and logging: The transaction protocol and logging component handles the exchange of messages required for two parties to sign electronic contracts.

The approach explored by the transaction protocol and logging component is to construct transaction protocol messages in such way that the identities of the involved parties, the causal relationship, and information describing the current transaction are bound together using cryptographic techniques. The purpose of the transaction protocol and logging mechanisms is to achieve nonrepudiation and increased confidence in that the information presented to a third party arbitrator reflects the actual events that took place during the protocol execution.

Authorisation and access control: The authorisation and access control component called μ-CAP, handles authorisation and delegation of access rights between users of the system.

Central to μ-CAP's operation is the use of credentials with properties similar to capabilities, but with an extended semantics that simplifies

the process of revocation. In μ-CAP, each issued capability is valid for one-time use only, and its access rights are automatically revoked by the system once used.

μ-CAP can be used to do more than enabling users to delegate access rights between users. In electronic-commerce systems, similar functionality to that of μ-CAP, is required to delegate trust between principals, e.g., to transfer the rights to deduct money from an account held on a remote computer.

Network-level security: The network-level security component called Secure IP (SCIP), handles authentication of end systems and establishes end-to-end secure channels between pairs of hosts at the network level.

SCIP is designed for small personal computing environments in which individual users manage their own machines and use them to access remote services. The system is designed such that it can be easily extended and used by a larger user community in wide area networks.

The main modes of operation of SCIP are host to host communication, remote site communication and firewall protection.

4.3. Hardware architecture

The fundamental property of the Trust architecture is the division between a secure subsystem consisting of a smart card and terminal I/O system, and an untrusted component consisting of the main CPU, memory and other devices.

For the correct operation of the security module, it is essential that the security module controls the communication of the input/output devices, the interaction with the (optional) smart card, and the communication between the security module and the local operating system. The transition between secure and insecure mode of operation is also essential to the operational behaviour of the system.

Figure 4.1 shows a schematic overview of the architecture. In it, a *trusted computing base* (TCB) comprising the following components are distinguished:

(1) a *security module* which supervises the actual signing and performs signature verification;
(2) a *smart card* which contains, protects and applies a signature key and stores the certificates that authenticate the key;
(3) a *log device* which is used as an append-only non-volatile memory for countersigned contracts;
(4) a *display device* and an *input device*.

For simplicity, is is assumed that the input device is a keyboard, but there is no reason for it not to be something else, for instance, a stylus for writing on screen.

The security module is connected to a *local computer*, and the local computer, in turn, will normally be connected to remote destinations via computer networks. The local computer is not part of the TCB, but it is expected to be the place where users run their software and where the initiative for entering into a contract and the negotiations and preparations for contract contents take place. Only when the contract is ready for signing does the security module become involved.

When there is no contract-signing in progress, the display and keyboard are available to the local computer as its input/output device. An X Windows server could be running on it, for example, with a Web browser in one of its windows. But, whenever a contract must be signed and/or verified, the security module temporarily disconnects display and keyboard from the local computer and takes them over for its own secure use.

In the following, the other subcomponents of the architecture are discussed in more detail.

4.3.1. Security module. The security module implements the core of the security subsystem and provides a secure interface to the local computer system. In particular, it provides an interface to the frame buffer memory area in the local computer system. The security module can issue read requests to the frame buffer memory with exclusive access privileges, and without being interrupted by other devices in the local computer system.

The exclusive access property mentioned above is essential to the operation of the security subsystem, because an atomic snapshot of the frame buffer contents can then be made.

The security module coordinates and executes the protocols for signing contracts and verifying incoming signed messages and contains the following three submodules: the *verifier*, the *signer*, and the *combiner*.

Verifier: The function of the verifier is to perform signature verification and construct credentials from certification chains in such way that the result can be presented to the user.

Signer: The function of the signer is to involve the user in signing statements. Before a statement is signed, it is displayed on the screen. A statement, therefore, is really an *image*, encoded in a standard form for others to display too. The signer locks the display before obtaining the user's permission to sign so that the user can read the statement. When the user approves, he or she presses an *accept key* and a PIN. The signer then gives (a digest of) the statement to the smart card for signing.

Combiner: The combiner allows a contract to be signed by the signer and the countersigned contract to be verified by the verifier, while making sure that the countersigned contract is identical to the one that was signed. The combiner also maintains state on outstanding contracts—contracts signed but not yet countersigned—and raises an alarm if countersigning is taking an unreasonably long time.

The security module also controls the input device, e.g., keyboard events. It can block some of these from being forwarded to the local computer system. The purpose of blocking events is to prevent that authorisation codes and passwords (PINs) are forwarded to the local computer system.

In addition, the security module can contain secure storage facilities for encryption keys and hardware support for encryption. This functionality can also be provided by a smart card. In the latter case it is sufficient that the security module provides a card interface.

4.3.2. Input/Output devices. Information presented to the human user is rendered on the system's output device. The image rendered on the output device is a visual representation of the current contents of the frame buffer memory. This visual representation is also the interface between the human user and the machine representation of that information. It could, for example, be information representing the state of an electronic transaction as perceived visually by a human user.

The terminal input and output devices are, when in secure mode, controlled by the security module. In that mode, window digests of display contents can be calculated. In the following input/out devices and window digests are described further.

Input device: The input device monitors responses from the human user and is controlled by the security module. Not all of the responses from the user are forwarded to the local computer system. In particular, the input device implements an activator for the secure mode of operation implemented by the security module (hard-wired), which is not forwarded to the local operating system.

Output device: The output device is assumed to be a pixel-oriented visual representation of the frame buffer memory. No semantic meaning is associated with the actual contents. It is, however, assumed that information can be rendered to its original displayed form as received from a peer application on a remote computer.

Window digests: Window digests are the hashed values of display contents, and can be calculated by applying a secure hash function to the pixel values. The pixel values of a window are represented as an octet string including their leading and trailing zero bits.

If H is a collision-free secure hash function, then it is infeasible to find another octet string that hashes to the same value. That is, when a collision-free secure hash function is applied to the octet string representing the window, it is computationally infeasible to find a different set of pixel values (for example, a window with different window content) that will hash to the same digest.

When computed with e.g., the MD5 message digest algorithm [Rivest, 1992], a window digest $H(w)$ of the window can be calculated and yields a unique 128-bits octet string. Modification of any pixel in the window results in a new and unique digest.

$H(w)$ can be considered to be a unique identifier for w. The strength of this scheme depends on the properties of the hash function used (for more details about this requirement, see Section 2.2.4).

4.3.3. Smart card. In the Trust hardware architecture, an optional smart card can be used to store the private key of the user, and provide digital signature functions. Private keys should never be revealed to any other component of the

system, and an authorisation code is required to activate the card for signing purposes. The authorisation code is never revealed to the local or remote computer systems.

Since smart cards typically have limited storage capacity, attention must be paid to how information (e.g., transaction logs) are transferred from a smart card to another computer system. The smart card can be equipped with a protocol that allows it to off-load its contracts into a trusted machine (the identity of which must be indicated by the user). Crypto-paging [Yee and Tygar, 1995], a technique where encrypted contents is transferred from a smart card to a remote computer, can be used to achieve this.

4.3.4. Local computer system. The local computer system component is a stand-alone computer system, e.g., a personal computer system with network access. No further assumptions or requirements are being made about its functionality except that it hosts the security module as its co-processor and is modified to interact with it to provide input/output access for the secure mode of operation.

4.3.5. Remote computer systems. Remote computer systems host parties that can run peer applications for contract negotiation. A peer application acts on behalf of a named principal that can be authenticated via one single third party authority, or a chain of third party authorities.

4.4. Secure mode

Transitions between secure and insecure mode of operation are performed in several steps. A transition is signalled either by the human user or by a software interrupt in the local computer system. The result of either of these events is that the security module enters secure mode of operation and can generate digital signatures (optionally using a smart card). When a mode transition signal is received by the security module, the contents on the display is maintained, but updates from the local computer are prevented. This guarantees that information remains static on the display so that the user can read it before signing. The mode transition is indicated to the user on a channel which is not accessible to the local computer. This could be a visual indicator on the output device (e.g., an LED). The indication can never be given when the security module is operating in insecure mode.

The next step is to obtain the user's authorisation code, which is a secret shared between the user and the security module (or smart card), and needed to activate it for signing purposes. The PIN request is indicated on the output device.[3] To prevent the local computer system from signalling the user to type the PIN while the security module is not operating in secure mode, the user must explicitly press a designated PIN key before the PIN is entered. If the security module is not already operating in secure mode, this key press will force it into secure mode. The net result is that the local computer will not be able to obtain the PIN.

[3]The indication could be another LED or a message rendered on the display.

The display contents and the PIN is forwarded to the security module. It verifies that the PIN is correct, and signs the display contents. The resulting digital signature is forwarded to the local computer.

If the local computer has obtained signed information from a merchant, the security module can verify its validity. The security module looks for a chain of certificates that authenticates the signature. It does this in interaction with the local computer. The information is shown on the output display, and the contents are forwarded to the security module together with the certificates needed to verify the signature. When the proper verification of signatures has successfully taken place, it is reported to the user.

Since the keyboard and display control is transferred from the operating system to the security module when secure mode is activated, it is unlikely that secrets will be revealed to the operating system. This, of course, assumes the user to be knowledgeable enough to only type his PIN when the system is in secure mode, and that this mode of operation is indicated visually. It is, therefore, essential that the interface to the security functions is comprehensible to lay users.

Although, in this particular design, computer, screen and keyboard can be packaged in a single device, there are no direct logical[4] connections between keyboard and display on the one hand and the rest of the computer system on the other hand. These connections can only be made via the security module and through the security module itself.

The following scenario illustrates the use of the secure mode of operation in conjunction with contract signing protocols. The participants are Alice, the client, and Bob, the merchant.

Alice must press the *Accept* or *Reject* key before typing her PIN. The idea is that Alice will quickly get used to pressing this key before typing her PIN—the system doesn't work otherwise. The two keys are guaranteed to place the security module in secure mode so that, anything typed immediately after pressing one of them will never be exposed outside the TCB.

Whether Alice should type a PIN, sign her name on the display with a stylus, have her fingerprint read or her retina scanned is an issue that is not addressed here. However, the architecture supports all of these solutions securely.

Another point worth making is that Bob, in our example, has nothing to do with Alice's PIN or with the method by which Alice's computer verifies that PIN. Bob relies on a set of certificates via which an authority trusted by Bob declares that Alice's signature can be relied upon. This stands in sharp contrast to current bank policies where the bank provides Alice with 'her' signature and PIN.

The contract displayed in a window on the screen is what Alice signs. The window contents must be coded as a bit string in a device-independent, standard way so that the contract on Alice's screen yields the same bit string as that on Bob's screen. What is actually signed is a secure hash of the bit string (and other, shared data) and this, of course, must also be a standardized hash function.

[4]Connections carrying logical signals as opposed to connections for supplying screen and keyboard with electricity.

It is possible to construct ASCII messages of a dangerous nature in such a way that, when they are interpreted as *images*, look completely innocuous. Alice could be presented with such a message and she could, therefore, be made to sign fairly arbitrary things.

This, fortunately, is a problem easily solved. The secure module attaches a *type*, as well as the dimensions of the image representing the contract to the message that is hashed and signed and written to the log. Alice's signature should be interpreted as a signature of the screen image specified by the message, not merely as a signature of a random bit string.

It is also possible that Bob hides information in the image, for instance by using low-contrast, very small characters. This is really the same as adding 'small print' to conventional paper contracts. If the print is so small that it is obviously meant not to be noticed, a judge will declare that part of the contract invalid.

Contracts must be time stamped in such a way that the time stamp, together with the identities of Alice and Bob, form a unique identifier for the contract. Alice and Bob must, therefore, also agree on the time stamp and the security module must verify whether the time stamp is recent and the identifier formed with it is unique.

The credentials that authenticate Alice to Bob and Bob to Alice form part of the contract that gets signed. They identify the parties who signed the contract and, as such, they form essential information for arbitration purposes.

4.5. Adapting existing technologies

The vital property of the architecture is that the TCB—which consists of security module, smart card, log store, display and keyboard—is not user-programmable and has been built by a reliable manufacturer. This, nevertheless, leaves a great deal of design freedom for specific implementations. This section explores some of the possibilities.

The Trust architecture uses a smart card, separate from the security module. If the two were integrated, the system would not necessarily be less secure, but, in many scenarios, the smart card gives greater flexibility. When smart card and log-device are removed from the rest of the system, what is left is a machine that is no longer personalized. In other words, somebody else can use it too (by providing their own smart cards).

The smart-card approach also makes it possible to switch from one workstation to another without being forced to change signature as well. However, there may be situations where it is reasonable to integrate smart card and security module, for instance in special-purpose devices tailor-made for making payments.

Much can be said about integrating smart cards and log devices. They could be packaged together as a single PCMCIA device, for example. However, the storage capacity for a log device may be so small that it fills up before the functional lifetime of the secret key in the smart card expires.

The simplest configuration is one that is not user-programmable: the functionality of the security module could be added to X terminals [Scheifler and Gettys,

1986], or to many of the now popular electronic 'personal organizers'. The contracts to be signed by such devices should either be prepared elsewhere (which will be the case anyway with most contracts representing railway/theatre/airline tickets, cheques, car-rental contracts, purchase orders, etc.), or by a local application that allows a window to be pointed at and turned into a contract.

It is more complicated to integrate the security module with a general purpose computer, especially if one does not want to modify typical PC architectures. The security module must be inserted in the path from processor to display and keyboard. With the keyboard typically directly integrated in the mother board, and a wide range of different graphics cards in existence, this is hard to do, but not impossible.

It makes most sense to design and build a new generation of graphics cards that integrate the traditional functionality of such cards with that of the security module. Keyboards then have to be routed via this card. At the same time, the keyboard can be equipped with slots for smart card and log device.

The smallest device considered is the *secure purse*. The secure purse is the size of a pocket calculator, has a low-bandwidth communication interface to offload payment information (including contracts and logs). It has a limited storage area for transaction logs. The device is not user programmable and is completely in the TCB.

A smart card can slide into a slot at the top of the purse. The purse contains the secure keyboard, and display as well as security module. The secure purse extends the interface of the smart card and provides a secure keyboard and display that can be used to authorize transactions securely, and not via any untrusted keyboards or displays.

The secure Personal Digital Assistant (PDA) is slightly bigger than the secure purse, but can be used to conduct more complex tasks. It has standard pre-loaded applications such as diary, spreadsheet, address book, etc. as well as payment applications. The principles of operation are the same as for the secure purse, but due to better display and supporting applications it will be more useful. The PDA may have wireless communication facilities to communicate with remote services and offload information, but may not allow program downloading. Some of these devices may have a slot for upgrades (e.g., memory upgrades), and a smart card may be fitted into this slot. Also the secure PDA (including its operating system and applications) is completely in the TCB. The main difference between the secure purse and the secure PDA is that the latter can render more complex information on its display, and assist the user with better application support.

The secure terminal PDA is a computing device consisting of a graphics display, keyboard, security module, and a slot for a smart card. A computer system with general computational resources is accessible via a computer network. Technologies such as the PARCTab [Weiser, 1991] can be redesigned to incorporate the security design. The information rendered on the display is controlled by a process located on a trusted terminal server in the network. The device renders information as received and forwards input events generated by user interaction back to the process

on the remote server. The security module can control the keyboard and display with only minor modifications to the rest of the system. The key issue, however, is that even though a remote process generates the pixel values to be rendered on the display, the security module embedded in the device itself fully controls the behaviour of the display and the screen. It is sufficient to provide means to freeze the display and its contents for signing purposes. Storage space for transaction logs and keys is limited to what the smart card offers.

The usefulness of the Trust architecture is best seen when integrated into standard personal computers. These systems run multitasking operating systems, and the user can download and install arbitrary new programs, including active programs and agent technologies as implemented by most Web browsers. New code is automatically downloaded to improve the user interface or to add new functionality (e.g., a new payment protocol implementation) and poses a risk to the user interface integrity.

Standard personal computers can benefit from the security design by the use of additional hardware that implements the security design. The design can be incorporated as a card attached to one of the system buses. Most important is that the device can detach the channels from the keyboard and display to the local computer system reliably, and that it cannot be controlled by the operating system. This includes that it cannot be interrupted when it accesses other system resources.

For example, if a PCI bus is used and the frame buffer memory area is known, the security module can be implemented as a PCI card that claims exclusive bus mastership whenever secure mode is being requested. It is important that the security module can obtain bus mastership, since otherwise there is no way to guarantee that the display contents is not being altered by other devices, and that input and output is not being monitored by the operating system.

The use of the security design in conjunction with standard personal computers yields much more flexibility than the other devices mentioned above. However, great care must be taken that the interface between the hardware implementing the security module and the personal computer operates correctly. The operating system can be used to run a transaction protocol while the security design generates the necessary digital signatures. For the same reason is storage capacity much improved. The operating system can save information encrypted by the security module by paging it to the hard disk (see also [Yee and Tygar, 1995]).

4.6. Summary and conclusions

This section discusses relevant issues of the Trust architecture. The focus of the discussion is on the human-computer interface, and on how to integrate the proposed design with existing payment protocols.

Humans base their decision on information presented to them, e.g., on the display of a personal computer or on ATM terminals. To ensure that the user makes a decision based on correct information, the integrity of this information must be verified.

The Trust architecture increases the level of security that is achieved when electronic transactions are conducted between mutually untrusted parties. Lack of integrity in the operating system and applications running on the computer does not affect the security achieved even though they are used to transfer application-specific information.

The main property of the architecture is that requests from the local operating system cannot disable the secure mode of operation, and that the main security functions have been isolated from the rest of the computer system. The effect is that the human user is kept in the control loop of authorisations.

No transactions can be authorized unless explicit actions are taken by the user. Security properties have, thus, been incorporated into the human-computer interface. The remainder of the chapter showed how the Trust architecture can be integrated into existing computer systems.

This chapter argues that it is essential that the user interface is under control by a secure process during critical operations, and that it is not acting on behalf of untrusted subjects. In addition, since the information shown on the display is what the user responds to, it is also a representation of the information the human user agrees to (e.g., when participating in a transaction protocol). The issue, thus, is to preserve the integrity of this information both during sign and verify operations.

Since it is assumed that sign and verify operations using digital signatures are in use, these operations must be tied to the information presented to the human user securely. In the Trust architecture, this binding is achieved by performing sign and verify operations on the pixels shown on the human-visible computer display.

Nonrepudiation is also an issue. In this respect, it is of importance to protect signing keys from disclosure. Since signing keys are protected by either a smart card or the security module itself, they cannot be disclosed easily to the host system.

The mechanisms described in this chapter implement a trusted path between a human user and the TCB, that is used to issue authorisation statements and verify displayed information. However, these functions still need to be integrated with other system components. In particular, the functions must be integrated with a transaction protocol. Thus, other issues have to be resolved. In particular, the interface between the proposed design and applications executing transaction protocols is essential.

—5—

Electronic Transactions and Logging

This chapter discusses the use of transaction protocols and logging facilities in electronic-commerce applications. The focus is on protocol requirements and specification. An example protocol for contract signing is also presented.

In electronic transaction systems, each participant should be provided with secure logging mechanisms. In the event of a dispute between the participants or a failure caused by the system, logged information can indicate what went wrong, and whether the failure in question was intentional (e.g., attack or fraud) or unintentional (e.g., a fault in the system). Although the principle of logging is fairly simple, only few applications yet exist that contain such functionality.

If all participants in a protocol are equipped with (secure) logging mechanisms, the asymmetric one-sided use of transaction logs present in some existing systems, e.g., in most cash-withdrawal systems currently provided by banks, is eliminated. The significance of multiple copies of logged information (for example, symmetric logging), is that in resolving a disputed transaction, each participant does not have to rely on computer-generated evidence gathered and presented by one of the opponents in the dispute.

In addition, independent copies of transaction logs protected by individual parties of a transaction increases the confidence in the integrity of the information. If transaction logs are gathered by one party only, the other parties of a transaction must trust this party not only to be honest, but also to implement correct logging mechanisms that are fair for all involved parties.

The rest of this chapter is organized as follows. First, the rôle of transaction protocols and of logging are described in more detail. Following is a description of requirements for secure transactions and for logging and a description of an example protocol developed for the Trust architecture. Finally, the chapter discusses some of the relevant issues and summarises the findings.

5.1. Overview

Electronic-commerce transaction protocols bind together client requests for services, the merchant's provision of these services, and the method of payment agreed upon by both the client and the merchant.

Some issues related to electronic commerce are similar in nature to issues in non-electronic commerce. In particular, in both electronic and non-electronic commerce, clients purchase services or goods from merchants and pay either directly using cash,

or indirectly using a trusted third party that vouches for the payment, e.g., a bank or credit card company.

However, the use of computer networks for electronic commerce removes one important property present in non-electronic commerce. In non-electronic commerce, the parties involved are physically located at the same place at the same time, or generally, have established a business relationship prior to the execution of a transaction.

In contrast, electronic transaction protocols are based on the exchange of messages via computer networks. Due to this distributed nature of the target environment, it becomes difficult to sign an agreement or contract between parties.

In open networked environments, electronic transaction systems are vulnerable to all the threats to network security discussed in Chapter 3. To counter these problems, authentication is required to determine the origin of messages, integrity checks are needed to detect messages that have been tampered with, and confidentiality is required to guarantee that a message is received only by the intended recipient. All this can be achieved with authentication and key distribution protocols that establish secure communication channels between communicating parties.

Existing protocols such as the Secure Socket Layer [Freier et al., 1996], Secure Hypertext Transfer Protocol (S-HTTP) [Rescorla and Schiffman, 1995] and Secure IP (SCIP) (see Chapter 7), handle some of the problems of insecure networks by establishing authenticated and confidential communication channels between the client's computer and the merchant's computer. However, the security provided by these protocols is clearly not sufficient for the purpose of electronic commerce. Network security is mainly concerned with secure communication over insecure networks, and often the communicating parties have established a trust relationship prior to the communication. In electronic commerce the main issues are the denial of having sent a particular message, or refusal to complete a protocol, and that the communicating parties may be mutually distrustful.

Fraudulent behaviour of either of the parties participating in a transaction is a concern. For electronic commerce, thus, the threat model is different than for the traditional network setting, and this advocates different solutions.

5.2. Arbitration

A distinction can be made between arbitrated protocols and adjudicated transaction protocols (see, e.g., [Schneier, 1996]). In arbitrated protocols, an independent arbitrator trusted by both the client and the merchant is used to complete a transaction. The use of an arbitrator is useful when the client and merchant are mutually distrustful. In non-electronic commerce, lawyers or notaries public are often used as arbitrators. However, the use of either lawyers or notary public in electronic commerce is more difficult, especially since their presence during the execution of transaction protocols may be required. The alternative is to use adjudicators: third parties that are involved as arbitrators only when a dispute occurs.

Due to the inherent cost of explicit arbitration of each transaction, adjudicated transaction protocols are built using two subprotocols, one non-arbitrated protocol

and one arbitrated protocol. The non-arbitrated protocol is executed every time a client and a merchant want to engage in a transaction. The arbitrated protocol is only executed in the event of a dispute over the transaction. An adjudicator then performs the necessary arbitration to complete the protocol (in some instance by the force of law).

Ideally, a transaction protocol should be self-enforcing and requiring no arbitration at any point. In practice, however, it is difficult to devise a protocol that is self-enforcing under all circumstances, and is supported by every legal system.

In electronic commerce, the use of arbitrators imposes several problems with different characteristics from those in non-electronic commerce. In the following, some of the problems are described in more detail:

- It can be difficult to locate an arbitrator that mutually distrustful parties both trust to perform arbitration between them. In non-electronic commerce, an advantage is that some people are looked upon as authorities on such matters. Also, direct face-to-face communication makes the process of finding a mutually trusted arbitrator easier, but an arbitrator is still required.
- The process of arbitration is likely to add additional costs to the amount that will be charged for each transaction. This cost may be partially due to increased network communication, but mainly due to the costs of the service of a third-party arbitrator. Legal advise, for example, is not considered to be cheap.
- An arbitrator can become the performance bottleneck in the system. Although replication services can limit this problem, scalability remains a concern.
- Arbitration introduces delays in transaction protocol processing. In the case of adjudicated transaction protocols a delay can also be introduced since the eventual outcome of a transaction is decided upon later, after the non-arbitrated protocol resulted in a dispute.
- Arbitrators are also vulnerable to various attacks in which an attacker may try to subvert it to gain benefits from disputed transactions. For example, denial-of-service attacks against an arbitrator, may severely affect the outcome of electronic transactions with timely execution requirements, e.g., in stock exchange and broker systems.

In the case of a transaction that results in a dispute, an independent arbitrator must be provided with sufficient information to perform arbitration and complete the transaction. Either the client, merchant, or both of them must reveal this information to the arbitrator for the arbitration process to take place. In the case of an arbitration protocol, this process takes place during the protocol execution. Arbitration is, thus, built into the protocol. For adjudicated protocols, the client and merchant must provide the adjudicator with information after the first subprotocol has completed. The information that the client and merchant provide to the adjudicator must properly reflect the actual events that took place in the first subprotocol execution.

Since the messages exchanged in a transaction protocol reflect the executed operations in the system, they, naturally, are the core of evidence of what actions actually took place, and they must be collected and stored securely. Both clients and merchants should be able to create and store an audit trail of all transaction-protocol messages arriving and departing from their machines.

If an electronic-commerce transaction system is to employ adjudication, it can best be reflected in the messages exchanged during the execution of first subprotocol. In addition, the design of an adjudicated protocol can be constructed to take into account that each party may want to gather transaction logs independent of another party.

5.3. Electronic transactions

This section describes the nature of electronic transaction protocols within the context of electronic commerce, a transactional view as seen from recoverable databases and audit trails and transaction logs.

5.3.1. Transaction protocol. In electronic commerce, transaction protocols are used for more than only the process of paying for a service. The entire process of requesting a service, the payment process, and to have the requested service properly serviced forms the transaction. In many respects the properties of electronic-commerce transactions can be compared with traditional transaction systems for recoverable databases.

The task of purchasing an item can be divided into different phases. This section describes in more detail the phases that takes place when a transaction protocol is used to purchase an item electronically. Three different phases are distinguished: *browse*, *purchase*, and *arbitration*. These phases are elaborated on below. The target environment in mind for this discussion is information networks, such as the World Wide Web.

A protocol execution is initiated when a user requests a particular service. The server in question is contacted, and a range of choices is obtained and typically these are rendered on the display for the user to make a choice. Information rendered on the display is assumed to contain enough information for the user to choose a service and to learn what the associated costs are. These attributes are visually confirmed by the user before a choice is made. In current implementations, software, such as World Wide Web browsers is used to render this information on the display. The information typically is generated on the fly by the server and returned to the client application for presentation to the user.

Phase 1, Browse: In the browse phase, the user accesses information from the merchant's computer via the client machine. On each request, information about the requested service is sent from the server to the client, and the client renders this information on an output device, e.g., as text or graphics on the display of the client machine. The browsing phase proceeds until the user finds an interesting service. He may then decide to proceed with the purchase of the item or service, or to cancel the current protocol without purchasing anything at all.

If the information contains a specific offer for a particular client, it can be electronically signed by the merchant computer either to limit its validity in time, or to whom it is destined. The user and the merchant may already have established a business relationship prior to the browsing phase, or they may be unrelated to each other at the time of the current transaction protocol execution, or introduced through a third party, e.g., with authority within their domain.

Phase 2, Purchase: In the purchase phase, the user proceeds and initiates a purchase request. This phase of the protocol binds the promise of payment presented by the user to the promise of being serviced by the merchant. In order for a purchase to take place, the exchange of some meta-information is necessary. Typically, the proper name of both the user and the merchant, and the credentials needed to authenticate them mutually would have to be exchanged. In addition, an agreement of a price and a method of payment is exchanged. When both parties agree, the purchased items are sent from the merchant to the user, and payment is sent from the user to the client. Either of these events may occur off-line. That is, physical items may have been purchased, and the method of payment could be plain cash.

Phase 3, Arbitration: In the case of a disputed transaction, phase three is required. If this situation should occur, a third party must determine the outcome of the dispute which is either to enforce the transaction in question, or to reject it. The arbitrator must base its decision on information collected during the transaction execution in question, and provided by either the user, the merchant, or, preferably, both of them. This information typically consists of logged information, receipts, or other evidence that may be used to determine the events that took place during the transaction protocol execution. A prerequisite is that the relevant information has been recorded by the system at the time the disputed transaction was executed.

5.3.2. A transactional view. Protocols used in electronic-commerce can, to some extent, be compared to transactions in recoverable database systems. In recoverable database systems transactions consist of groups of operations that the system should perform atomically. In a typical transactional environment, one or more clients execute a transaction protocol with a server acting on behalf of a database service. A client begins a transaction with a *BeginTrans* request indicating the start of a transaction to a server. The client then issues the operations that should be performed atomically, and finishes the transaction with an *EndTrans* request to the server. If the transaction progresses without any error, the server informs the client that the transaction has committed. Otherwise, and in case of a failure, the transaction is aborted and has no effect.

Transactions protocols, as used in electronic-commerce applications, share some of the properties of the traditional transaction model used in recoverable database systems. In particular, the all-or-nothing atomicity semantics is similar; unless a

service is being paid for, it should not be delivered either. In [Härder and Reuter, 1983], the term ACID is used to describe the other fundamental properties of transactions in database recovery:

Atomicity: A transaction must have all-or-nothing semantics. It either completes successfully and its effects are recorded in the data items, or it fails and has no effect on the data items at all.

Consistency: A transaction takes a system from one consistent state to another consistent state.

Isolation: The intermediate effects of a transactions are not visible to other transactions.

Durability: All effects of a transaction that have completed successfully are stored on a permanent storage device.

In [Black, 1991], ACID properties are used to describe transactions within the context of operating systems, and it is being argued that operating system services based on the transaction model increase system reliability. In [Coulouris et al., 1994], it is argued that the rôle of the consistency property mentioned above should not be included in the list of properties above. It is argued that it is generally the responsibility of the programmers of servers and clients to ensure that transactions leave a database consistent, and therefore, the consistency requirement should be considered optional among the ACID properties of Härder and Reuter.

In electronic-commerce applications, employing transactions protocols, however, the rôle of the consistency property may be different than for recoverable database systems.

There are several similarities between traditional transaction systems and transactions in electronic-commerce systems. Clearly, electronic-commerce transactions must satisfy at least some of the ACID properties stated above. For example, in electronic commerce the outcome of a transaction may be recorded as not successful. Still, it is valuable to keep information about the operations that were carried out unsuccessfully. This property may be useful in the case of arbitration. Note that in traditional recoverable database systems similar accounting and cleaning functionality is required.

The *atomicity property* is as important for electronic-commerce transactions as it is for transactions in recoverable database systems. However, the reasons for this importance are somewhat different for the electronic-commerce transactions: The atomicity property is concerned with the binding of a specific payment to a specific provided service. Preferably, the user should receive the requested service, and the server should be paid for it. Either none or all of the operations should be carried out; this is particularly desirable in contract signing between two or more parties. Arbitration is useful if atomicity cannot be achieved.

Note that a computer system cannot always monitor and verify that a transaction finishes satisfactorily. When, for example, physical goods or information are delivered as part of a transaction, humans are the only principals capable of judging whether the obligations of a transaction have been met. Thus, third-party arbitration is unavoidable, since disputes concerning e.g., quality of delivered goods

cannot be assured by the means of a protocol. In particular, the process of arbitration is going to take place off-line, and not as part of a real-time transaction protocol execution. Consequently, the transaction protocol should, by itself, exchange sufficient information to enable an off-line arbitration to take place based on the message contents, at some time after the protocol has finished.

The *consistency property* states that a transaction takes a system from one consistent state to another consistent state. It was mentioned above that the importance of this property in recoverable database systems is somewhat controversial. However, for electronic-commerce transactions, the property reflects the correctness of the transaction protocol, e.g., the client and server software that implements the transaction protocol in question. Although the correctness of the implementation is essential, the argument for the consistency property remains the same as for traditional recoverable database systems.

In recoverable database systems the *isolation property* states that intermediate effects should not be visible to other transactions. This property is mainly concerned with serial equivalency of concurrently executing transactions. This may be of concern to some specific transaction protocols, but not in general.

The *durability property*, supporting the permanence of successful transactions, is of importance for electronic-commerce transactions—especially in the case of unsuccessful transactions. The storage of transaction data is fundamental in the cases of disputes, nonrepudiation, or any failure that might occur in the system.

5.3.3. Auditing and transaction logs. This section explains the rôle of audit and transactions logs, and the requirements for proper logging of transactions in an electronic-commerce system. Computer generated logs have been used as evidence in computer crime lawsuits (see Section 5.3.4). Computer generated logs may also serve a purpose in electronic commerce by providing transcripts of protocol executions.

Auditing is a method used to record the events that occured in a system. Transaction logs play a similar important rôle in recording events occuring in electronic-commerce transactions. Computer generated auditing logs are often the information used for arbitration or legal action purposes.

Since transactions may be conducted between unrelated parties where one or more of the involved parties can be dishonest and not adhere to the given protocol, it is important that the created logs reveal any such attempt of fraud to the other participants either implicitly or explicitly. It is also important that what actually constitutes an agreement between the involved parties is clearly defined, and part of the protocol definition.

Logging of information is essential both to the recovery of failed transactions, and to perform arbitration of disputed transactions. In many other areas, logs have been used successfully to record the truth about the state of objects. Logging is a reliable and cheap method of recording such information, and the use of logs is considered good engineering practice [Lampson, 1984]. Logs are append-only data structures. Since a log is a simple data structure, it can be read, written, and

backed up on stable storage devices in a fast and straightforward manner. These are also the reasons why logs are currently being exploited to build fast file systems.

It has been pointed out elsewhere that write-once hardware devices can support the append-only process of logging by better protecting the logged information [Mullender, 1996]. Furthermore, the correctness of logs is generally easy to verify, which makes it a good candidate for auditing of transactions.

5.3.4. Logs as evidence. The use of computer generated logs as evidence in computer crime lawsuits has been extensively studied. Log contents are often classified as "hearsay" material. In addition, since ordinary logs can sometimes easily be modified by their owners, their use as evidence in court poses some interesting implications.

Logs are essential for the process of detecting that something went wrong, and are at best, the only information left after an incident has occured. In [Cheswick and Bellovin, 1994] the use of computer generated logs and the legal implications of presenting them as evidence against hacking activity in the United States are discussed. Generally, the following advice for the use of computer generated logs:

(1) To qualify as evidence, a log must be created reasonably contemporaneously with the event in question.
(2) The system must have been designed for logging. Since logging is carried out by a computer program, a priori knowledge of how proper logging is performed, must be required from programmers and system administrators.
(3) Logs must be kept as a regular business practice. Random compilations of data are not admissible.
(4) From a legal point of view, it is advantageous if the contents of a log is actually used for some daily procedure or purpose. This demonstrates the owner's faith in the correctness of the log, and increases the confidence in the integrity of its contents.
(5) A custodian or qualified witness must be able to offer a description of what system is used, where the relevant software and hardware components came from, and, in general, something about the accuracy and integrity of the logs.
(6) Strive to increase apparent validity of the logging functions. Software written by companies is more valuable as evidence than homegrown software.
(7) Logs stored on safe storage devices, e.g., WORM[1] disks, are less vulnerable than ordinary storage facilities. In addition, it is also prudent to store the logs in a different location than the machines on which they are generated.

Forging ordinary computer logs is relatively easy. Clearly, when logging is used as evidence in the context of electronic commerce, there is always a chance that users changed their own logs to gain some economic benefit. In the case of a

[1]Write Once Read Many.

disputed transaction, this, at least, is an argument that users will be faced with. Even if the log is authentic, sloppy routines may be used to discredit the user.

5.4. Logging transactions

In this section transaction logging is described in more detail. The main purpose of logging the messages exchanged in a transaction protocol, is to create a record of what each party did during the protocol execution. In some instances, this requires that other meta-information generated during the protocol is also put into the log, e.g., retransmitted messages. The transaction protocol described below makes the causal dependency between the individual protocol messages explicit, and contains sufficient information to identify the involved parties and information exchange.

5.4.1. Transaction-protocol requirements. Transaction protocols are similar to the more traditional authentication and key distribution protocols in that the design of such protocols should follow similar design principles and rules. In particular, explicit security is of importance (see Chapter 3). The protocol requirements outlined below are similar to the fail-stop protocol requirements described in Section 3.7.3:

(1) Each message should contain the identity of its sender, the identity of its intended recipient, a protocol identifier and version number, a message sequence number, and a freshness identifier.
(2) Each message should be signed by the sender's private key.
(3) Each transmitted message should contain sufficient information to refer to previously received protocol messages. If all transaction messages are recorded in the log, the hashed value of the previous message is sufficient.
(4) Honest processes should follow the protocol and ignore all unexpected messages, and halt a protocol execution in which an expected message does not arrive within a specified interval.

In comparison with the original fail-stop protocol description, the main addition in the list of requirement presented above, is that each transmitted message properly reflects communication prior to that message. Note also that in the original description of fail-stop protocols, each message is optionally encrypted with the public key of the intended recipient. Encrypted messages can only be read by principals with access to the required private keys. This reduces the usefulness of such encryption for logged messages.

5.4.2. General logging requirements. For general logging of electronic transactions, the following technical requirements are identified:

(1) Each message sent or received as part of the protocol should be written to the transaction log immediately upon reception or transmission.

    Logging can therefore be considered an extension of transaction protocol processing functionality, and be designed as part of the transaction system both for client software and for merchant software.

(2) Logged information should be kept in a format that encourages ease of use, e.g., as being readable by plain text editors. At least, the user should be able to observe a view of the log of the transactions that took place. This need not consist of the exchanged messages, but could rather be the information found in the log after it has been formatted by a filter or a pretty printing program. Note that functions will be required to display signed messages in this way.
(3) Logs should be kept off-line, and not on the client computer used to carry out the actual transaction protocol. This is not required per se, but as it was argued in Section 5.3.4, logs kept either off-line, or on a different computer that the one where it was generated, can increase confidence in its contents (e.g., by using external WORM devices).

When an append-only logging operation is combined with cryptographic functions to chain information, it is an ideal approach to protect electronic transaction records. A secure transaction log should satisfy the following requirements:

- The causal relationship between messages should be clear in the log. In order to achieve this, each message must contain sufficient information to express the causal relationship with regard to its predecessors. The requirements to the messages of the transaction protocols enforce this.
- Each message should be signed by its sender, and when received, not be put into the log of the recipient before the recipient has verified the attached signature.
- The encoding of a log entry must be constructed in such way that a third party in the same domain and with knowledge about the public keys of both the service and the user can verify that it indeed is an agreement established between them. Both parties shall also contribute to the contents of the log in such way that its encoding shows that both of them agree to the semantic meaning embedded therein. The log is kept by either of the parties and disclosed in the case of a dispute.
- The acceptance of a message n by a recipient should be reflected in message $n+1$ by containing some evidence of this. A signed hash of message n may suffice. Note that this is not the same as using a nonce as freshness identifier.

5.5. Contract negotiation

This section describes a contract negotiation protocol that relies on the architectural components described in Chapter 4. In the following, the "characters" introduced in Chapter 4 are reintroduced throughout the protocol description.

Suppose Alice wants to sign an electronic contract with Bob. In the following the actions taken by Alice, her local computer, her security module and her smart card are discussed. Bob, in this example, is an entity at the remote end of a computer network. Bob could be a person with a setup like Alice has, or an organization, for instance, an airline with an automatic reservation and ticket-selling system.

5.5.1. Local computer. The initiative for entering into a contract can be Alice's, Bob's, or an application executing on either Alice's or Bob's local computer. In all cases, the initiative must be taken over by applications executing on Alice's local computer.

Alice's local computer makes all the preparations necessary for signing the contract. When these have been made, it will give the information relevant for signing to Alice's security module and requests it to take care of the actual signing and verification of the contract. More specific, the local computer takes the following steps:

- it negotiates, presumably in interaction with Alice, a draft contract, $Contr$, with Bob (or *his* local computer),
- it also negotiates an arbitrator, Arb, acceptable to both Alice and Bob;
- then, it exchanges with Bob the credentials,[2] $Auth_A$ and $Auth_B$, that, respectively, authenticate Alice's key K_A to one of Bob's certification authorities (CA), and Bob's key K_B to a CA trusted by Alice;
- it displays $Contr$ on Alice's display in a rectangle $Rect$
- and requests a signature from Alice's security module by passing to it

$$\{Contr, Rect, Auth_A, Auth_B, Arb\}$$

- when it receives Alice's signature, Sig_A, from her security module, it sends that to Bob; meanwhile,
- when it receives Bob's signature, Sig_B, it sends that to Alice's security module for verification;
- finally, it receives a success/failure notification from Alice's security module.

The steps that need to be taken in case of errors in the protocol have not been discussed so far. Alice can refuse to sign, Bob can refuse to sign, the network can break down, etc. Section 5.6 contains a more elaborate discussion of error handling and related problems.

5.5.2. Combiner. The combiner in Alice's security module, upon receiving

$$\{Contr, Rect, Auth_A, Auth_B, Arb\}$$

does the following:

- The combiner records the contract as *'not signed'* and *'not countersigned'*, and starts signature and countersignature timers. If the timers expire, there may be a problem getting the contract signed or countersigned.
- It gives $Auth_B$ and Arb to the *verifier* which generates the credentials for Bob, the counter-signer, and for the arbitrator.
- Then, the combiner offers the contract plus credentials to the *signer* in order to obtain Alice's signature;
- when Alice's signature has been placed, the combiner marks the contract *'signed'* and hands it to the local computer which sends it to Bob;

[2]Typically, chains of certificates.

- when the countersigned contract arrives with Bob's signature (this could happen before or after the previous step), the combiner checks that it is the same contract as the one (to be) signed and hands it to the verifier for verification.
- when the contract has been verified, the contract is marked *'countersigned'*
- as soon as a contract has been marked *signed* and *countersigned*, it is appended to the contract log and Alice is notified via a message on the screen.

5.5.3. Signer. Before discussing the failure modes, the actions of the signer and verifier are described. The signer carries out these steps:

- The signer turns on *secure mode*, that is, updates from the local computer to the display are refused and keystrokes are not passed on to the local computer;
- it then clears the screen and displays *Contr* in *Rect*.[3] The signer also displays the credentials for Bob and for the arbitrator so that Alice can check with whom she is entering into a contract and if the arbitrator is appropriate for the contract. These credentials are presented in the form of text. Alice can now read the contract and decide whether or not to have the security module sign that.
- Alice accepts or rejects the contract by pressing an *'Accept'* or *'Reject'* key and she types her PIN.[4] The signer can flash an LED under these keys to alert her to the fact that she is being asked to sign a contract. The security module will make sure that the LEDs can never be used for anything else.
- The signer sets $Type$ to *Accept* or *Reject*, depending on the key that was pressed;
- creates a message, M, consisting of

$$M = \{Type, Time, Auth_A, Auth_B, Contr\}$$

- computes a *secure hash* H of M, $H(M)$, and offers that, with Alice's PIN, to her smart card. The smart card verifies the PIN and signs the message if the PIN is correct;
- the signed hash, $\{H(M)\}_{K_A}$, together with M is returned to the combiner. From there, it will find its way to the local computer and to Bob.

---

[3]The only function of *Rect* is to allow the security module to display a contract in the same position the local computer placed it: it is a convenience factor.

[4]To prevent Trojan-horse software from luring Alice into typing a PIN to malicious software on the local computer, Alice must explicitly press an *'Accept'* or *'Reject'* key on the input device before typing the PIN itself. Alice will get used to this so she will learn to press one of these keys always before typing her PIN.

Depressing an Accept or Reject key when the security module is in normal mode will cause it to switch to secure mode and give an error message. It will not pass any subsequent key strokes—which may represent a PIN—to the local computer until a reasonable timeout period has expired.

5.5.4. Verifier. The verifier carries out several kinds of verifications. It manages a set of local certificates that are used to derive the credentials of the signatures on received messages. Certificates can be added to or deleted from the set; additions and deletions can only be made with authorization from Alice's PIN. It also verifies the signatures on incoming messages. Signature verification is straightforward:

- The verifier receives from the combiner a message,
$$M' = \{Type', Time', Auth_B, Auth_A, Contr\}$$
and a signed hash, $\{H(M')\}_{K_B}$.
- From $Auth_B$, the verifier derives that the message must be signed by K_B (encrypted with K_B^{-1}).
- The verifier checks the time stamp and makes sure the message is neither too old nor too new.
- It then calculates H' in two ways: by computing it from M' and by decrypting the signed hash with Bob's public key K_B. If the two are the same, the signature is verified.
- The verifier reports success or failure to the combiner.

5.6. Discussion

If something goes wrong during the preparation phase of the contract, that is, before the contract is offered to Alice's security module, the signing operation can safely be abandoned. The signing protocols are thus greatly simplified by doing all the certificate verification before signing operations start.

All the interesting error cases occur when one party signs a contract and does not receive a countersigned copy from the other. This could be the result of a refusal on the other party's part, or of a failure—benign or malign—in the system.

The simplest case occurs—at least, from Alice's point of view—when Alice simply refuses to sign the contract. She presses the *Reject* key, which causes her system to send a signed refusal to Bob. If Bob's already signed, he must store this refusal. Otherwise, Bob may or may not decide to keep it. Alice does not care; she has not signed the contract, she's only signed a statement that she's not signing the contract.

Without Alice's signed refusal, Bob must seek arbitration; with the refusal, Bob has a statement signed by Alice to prove that the contract is not valid, even though Bob did sign a copy.

Consider the case where Alice does sign her copy of the contract, sends it off to Bob and does not get anything back, not even a refusal. Alice's security module will detect this through time-out and it will display a failure message on the screen. It will also get the smart card to sign a statement to the effect that the contract-signing operation failed, append that to the log, and pass it to the local computer as well. As far as the security module is concerned, the signing operation has finished.

Alice's local computer, or Alice herself must now contact the arbitrator—let's call him Donald. If Donald is on line, she can contact him and give him a copy of the signed contract as well as a copy of the signed failure message. Donald will then try to contact Bob. If this succeeds, he can confront Bob with the contract

and ask him to commit to the contract by signing it, or commit to refusing to sign the contract by signing a statement to that effect. This statement is returned to Alice and appended to her log (this can be done in the form of a 'contract' with the adjudicator, so that Alice does not need to implement a separate mechanism in her security module).

If Donald cannot contact Bob, he files Alice's complaint in a time-stamped record so that the problem can be sorted out later. If Alice can not contact Donald on line, she can call him on the phone instead. Getting *Bob* on the phone may have a great deal of social use, but it cannot be used by Alice as protection against Bob's possible dishonesty.

Donald can carry out these interactions with Alice and Bob using the same contract signing protocols that Alice and Bob should have completed.

5.7. Summary

In this chapter logging of electronic transaction protocols was investigated as a tool to increase the level of security in electronic commerce. The usefulness of this approach depends on explicit and well-defined transaction protocol formats, and binding of protocol message contents.

The use of logging facilities increases the level of security in an electronic-commerce system. However, as much of the previous discussion has shown, its usability as evidence in a lawsuit may be limited. In fact, the use of computer evidence in court has shown to be rather difficult. Logging constitutes one of the simplest forms of computer evidence, and yet, may face legal problems.

Increasing the confidence in the logged information is what seems most important to achieve. Robustness and explicitness are the keywords to the enhancements of existing logging functions.

The argument for secure transaction logging can also be seen from a safety point of view. Although common sense says that logging functions should be mandatory in transaction systems, still few of these systems provide independent logging functionality.

The atomicity property of an electronic-commerce transaction cannot be enforced online in a way that guarantees that the client receives the service or goods, and the merchant the proper payment. This is also the case for systems employing trusted third parties. Consequently, the process of arbitration over disputed transactions must take place off-line at some time after the transaction protocol has finished. The system can support arbitration by providing transactions logs designed with arbitration in mind.

—6—

Authorisation and Access Control in μ-CAP

In a personal computing environment, one cannot always assume that all computers are online all the time. The enforcement of an access control policy should therefore not depend on a particular machine currently being online and reachable via the network. In such environments, the enforcement of access control policies needs to be carried out by servers, and only to a lesser degree depend on the accessibility of other remote services.

This chapter describes how authorisation and access control is achieved in μ-CAP.[1] The μ-CAP architecture is designed to let users define their own access control policies and have them enforced by remote servers. In μ-CAP, access rights can be propagated between users and exercised by a server without any communication with the object server prior to the particular access request.

Capabilities, in the form of delegation certificates with once-only semantics, are used to implement propagation of access rights. When a request to access an object is invoked, the associated delegation certificate is revoked, and thereby invalidating the delegated access rights. An underlying assumption is that object servers are trusted by object owners to authenticate and enforce received delegation certificates according to the current access control policies.

The once-only semantics of delegation certificates can also be used to implement electronic payment systems. Electronic payments are, essentially, statements of trust exchanged between principals, e.g., users, merchants and banks. μ-CAP provides a simple mechanism to transfer and enforce such statements.

This chapter is structured as follows. First, an overview to the problem is presented together with a short survey of related work on authorisation and access control. Following the introduction is a detailed explanation of the μ-CAP architecture and access request protocols. The explanation also covers the assumptions underlying μ-CAP's design, and the encoding used for μ-CAP's protocol messages and delegation certificates. Following the architectural description, an analysis of the access request protocols is presented together with a discussion on how μ-CAP can be used to implement electronic payment systems. The chapter ends with a discussion of the proposed design and a summary of the results.

[1] Details of μ-CAP's initial design were conceived in collaboration with Tage Stabell-Kulø, University of Tromsø, Norway. The first results were presented at the 1st DIMACS WorkShop on Trust Management in Networks [Helme and Stabell-Kulø, 1996]. An overview of the intended target environment, the distributed File Repository (FR), is presented in [Helme and Stabell-Kulø, 1997].

6.1. Overview

The introduction of mobile computers into distributed systems has lead to the development of new paradigms in distributed computing (for example agent based computing and applet programming). New paradigms often require changes or a total redesign of existing architectures [Badrinath et al., 1993, Mullender et al., 1995].

Mobile computers are often used as personal computers. When personal computers are integrated into distributed systems, they will be used to store personal information. Amongst others, users will store private information, encryption keys and the credentials needed for authentication and access to remote services. A user will, typically, own and use more than one computer, e.g., a workstation at the office, a personal computer at home, a laptop while travelling, and a palmtop or PDA as a personal organiser.

A consequence of using more than one machine is that personal information belonging to users is stored in a distributed fashion. Lack of network connectivity means that some computers may not always be online (e.g., a laptop or palmtop could be switched off) and therefore be unreachable to other computers during periods.

μ-CAP is designed to provide security services to the File Repository (FR), a distributed file system based on a version control access model [Helme and Stabell-Kulø, 1997]. File operations of FR relies on the security services provided by μ-CAP to enforce the security policies defined by the users of the system.

FR provides services for basic file operations and file sharing. FR is designed for mobile computing environments where machines sometimes can be switched off or not reached due to network failures. In FR, each file has an owner and can over time have many versions. Copies of files can be distributed over multiple servers, although one copy is always the master copy. FR servers provide storage for files, and the possibility to store state about a file's whereabouts. When a file is to be shared, the owner must grant access to the file (typically read-once or write-once access, or both). New file versions checked in by another user can then be committed by the owner.

μ-CAP aims at providing authorisation and access control functions for computing systems consisting of personal computers when incorporated and used in open networked environments. Central to μ-CAP's operation is the use of credentials with properties similar to capabilities, but with an extended semantics that simplifies the process of revocation. In μ-CAP, each issued capability is valid for one-time use only, and its access rights are automatically revoked by the system once used.

Another distinct feature of μ-CAP, is that individual users are in authority to generate capabilities for the objects they own that remote object servers store and protect on their behalf. Newly created capabilities can be propagated to any user. Users who receive these capabilities can present them to servers in order to request a service only once. The server in question revokes the capability upon reception of a service request made with it. These features of μ-CAP enable access rights to

be propagated with few assumptions made about the presence of online services to support the process of authorisation and access control.

μ-CAP can be used to do more than enabling users to delegate access rights to other users. In systems designed for electronic commerce, for example, similar functionality to that of μ-CAP, is required to delegate trust between principals. This, could be the rights to deduct money from an account held on a remote computer only when certain requirements are met. μ-CAP can in this respect be used as a simple electronic payment system where the delegation certificates represent signed cheques that authorise specific actions. For example, by issuing a signed cheque authorisation in the form of a capability with once only semantics, a user can effectively delegate to another user the rights to obtain a service once, or to deduct a specific amount of money from his bank account once.

6.2. Authorisation and access control

Protection and access control is concerned with the enforcement of security policies, and decisions concerning what principals have which access rights to particular objects. Research into the field of authorisation and access control is thoroughly described in literature (see for example [Jones, 1978, Wilkes and Needham, 1979, Landwehr, 1981, Wulf et al., 1981, Gong, 1989b, de Viva et al., 1995]).

The properties of personal mobile computers have particular implications for the design of authorisation and access control systems. More specifically, many existing systems facilitate the use of online services to implement security services. If online services are critical to the correct functioning of the authorisation and access control functions of a system, and network connectivity cannot be guaranteed at all times, applications with security requirements are penalized in such systems.

Clearly, for the design of an authorisation and access control system, the assumption that a machine can be unreachable, has implications for the design of security functions. In particular, the use of online services should be minimised.

In Kerberos for example, loss of network connectivity, brings the entire system to a halt and clients are not able to obtain credentials for any service until the server is online again. A slow link to an authentication server in Wide Area Network (WAN) settings can be annoying for local users as can be experienced when using the Andrew Distributed File System (AFS) [Morris et al., 1986] when the authentication server is located at a remote site. At the cost of more complexity the use of replicated services can reduce these problems but not eliminate them entirely [Gong et al., 1993].

Although it can be argued that current designs of authorisation and access control systems often are quite sound for traditional distributed systems, the use of trusted third party (TTP) security services is infeasible for personal distributed computing environments. For example, a user may want to delegate access rights for a particular file to another user. If this user is not registered in the same security domain as the file owner, it is difficult (or even impossible) to create the credentials that a file server will accept as valid. There are no fundamental or technical reasons

why such restrictions should exist in the first place.[2] A user should simply be able to delegate access rights electronically to anyone without involving a third party. The main issues are that the file owner must provide the other user with valid credentials to access the file in question, and the system must be able to verify the validity of these access rights, and ensure that they cannot be misused. This argument is naturally valid also for other types of services.

Another important issue is the revocation of issued access rights. In centralized systems, the process of revocation is relatively straightforward. Since all channels on which access requests and propagated access rights are under centralised management and control, revocation mechanisms can be implemented effectively. In distributed systems, however, revocation is far more difficult to achieve. Decentralised control, unrelated organisational domains, and insecure networks, lead to different challenges than for the centralised case. In particular, fast end effective revocation of access rights in distributed systems depends on either the presence of online revocation services, or access rights that are limited in time. Either approach has its advantages and disadvantages. Effective revocation of access rights in distributed systems is generally considered to be a hard problem to solve [Lampson et al., 1992b].

There appear to be two main approaches to the design of protection mechanisms: access control based on either access control lists or capabilities, and an access matrix describing each principal's access rights to the resources available in the system [Lampson, 1971]. The main difference between the two approaches is that the former views the access matrix by its row, and the latter views it by its column. Capabilities [Dennis and van Horn, 1966] are held by the subjects requesting access and usually carry with them the right to access system resources. Access control lists contain names of authorised principals and are kept together with each object, and are maintained by servers in the system,

Some systems use access control lists (ACL) to express access policies. The most general ACL is a list of policy statements. Each policy statement can be simple or compound, and describe complex relations between principals in the system (see, e.g., the calculus for access control described in [Abadi et al., 1992]). Simple policies, for example, can express that one particular principal is authorised to access a particular file. Compound policies can impose further restrictions, e.g., as *rôles* [Nyanchama and Osborne, 1993], by stating that a principal can only access a resource when the access request originates from a particular program when executing on behalf of that principal (e.g., a user).

In contrast to ACL-based systems where access rights are held close to the objects, capabilities are held by the subjects in system, and presented when resources are being accessed. In traditional capability-based systems, the main security problems are that holders of capabilities can propagate them at will, and that whoever

[2]It should be noted that there may be politically motivated reasons, though. For example, a company might require that all of its employees are only allowed to encrypt information with keys shared with a central authority which is maintained by the company (for implications and a practical solution see [Blaze, 1994a]).

holds a capability can exercise its rights. In some capability systems, capabilities are identity-dependent and can only be used by their owners.

Securing a capability system requires breaking the chain of free propagation and free access. That is, limiting how and by whom access rights can be propagated between the subjects in the system. These problems can be solved using one of the following approaches [Gong, 1990]: (1) prohibit free propagation but grant a principal whatever access rights that are held in his capabilities, (2) prohibit free access but allow free propagation, and (3) prohibit both free propagation and free access by making capabilities identity dependent.

In distributed systems, authentication is fundamental to the correct execution of access control functions. Since all communication channels on which access requests can arrive are not under centralized control, authentication is essential to determine the origin of the access request. Cryptographic protocols [Schneier, 1996] are usually employed to provide secure channels between mutually trusted entities.

Trust is fundamental to authentication in distributed systems and many models for trust relationships in distributed systems have been proposed in literature (see, e.g., [Lampson et al., 1992b, Yahalom et al., 1993, Wobber et al., 1994]). Access policies are, essentially, statements of trust, and express trust relations between principals.

In the following, some of the systems that motivated the design of μ-CAP are discussed in more detail. In particular, the following systems are discussed: the Amoeba operating system [Mullender, 1985, Tanenbaum et al., 1990], ICAP [Gong, 1989a], PolicyMaker [Blaze et al., 1996b] and Simple Distributed Security Infrastructure (SDSI) [Rivest and Lampson, 1996].

Amoeba relies on capabilities to implement authorisation and access control functions. In Amoeba, all objects are identified by capabilities of 128 bits length. Each capability contains sufficient information to identify the server holding an object (*server port*), and an identifier (*object number*) for the object. In addition, each capability also contains access permissions and check fields to identify the types of access the holder of a capability is allowed to make. Amoeba's capabilities are "scrambled" to prevent forgery, and tampering yields an unusable capability.

In Amoeba, protection relies on the difficulty of creating forged capabilities that may represent access rights to particular objects. Random bits in the capability increase the sparseness of each capability in this respect.

Amoeba allows capabilities to be freely propagated. Revocation of a capability requires that the object referred to by a capability is unregistered, and a new capability is issued for the object.

ICAP employs the use of identity-based capabilities where the semantics of traditional capabilities have been changed to also incorporate the identities of capability holders. In ICAP, capabilities are issued by object servers. The server keeps an internal capability for each object, which is assembled from an object identifier and a verification code. External capabilities are issued for each subject exclusively, and contain sufficient information to describe a subject's access rights

to the object in question. An underlying assumption is that the server can authenticate each subject properly and determine whether it is the owner of the issued capability who issued an access request.

Propagation of access rights is implemented using an explicit handshake with the object server. A principal that wants to propagate access rights for an object forwards this request to the object server. The server creates a new external capability for the object containing the propagated access rights and the identity of the principal that access rights are propagated to. The new capability is presented to the intended principal, and can be used by that principal to exercise the propagated access rights in subsequent communication with the server.

The server keeps an exception list on a per object basis which is used to implement rapid revocation of access rights. For each granted revocation, the corresponding exception list is updated to reflect the revocation of the capability in question. On access requests, both the validity of the capability is checked, and the exception list is checked to prevent the use of a capability that is already revoked. When an exception list becomes to long, a full revocation can take place, and a new internal capability is issued for the object.

PolicyMaker is a certificate-based authorisation system. The system differs from other certificate-based authorisation systems in that instead of binding names to public keys, it binds public keys to predicates that these keys are trusted to sign. Using a simple programming language, each predicate can express policies, credentials or trust relationships between principals. An important aspect of PolicyMaker is that the mechanisms used to verify credentials do not depend on the credentials or the applications that use them. Therefore, PolicyMaker enables applications with different policy requirements to share the same verification infrastructure.

In PolicyMaker, security policies and credentials are expressed in terms of predicates associated with public keys. These predicates either accept or reject actions based on what the corresponding holders of secret keys are trusted to do. Whether an action should be accepted or rejected, is determined by queries to PolicyMaker.

The core functionality of PolicyMaker is to process queries, and the functionality of the system can in this respect be compared to a rule-based inference engine. Queries are requests to determine whether one or more particular public keys are authorised to perform some action, and whether this action is in accordance with a given policy.

Policy assertions are defined in advance and prior to system operation. In PolicyMaker, a policy assertion is either a certificate or a local policy. In contrast to certificates, local policies do not need to be signed with a public key, and are always valid locally.

SDSI is a simple public-key infrastructure designed to address several problems of existing public-key infrastructures. In particular, SDSI tries to address the incompleteness and complexity of other proposed systems (e.g., X.509, see Section 3.8) by being simpler, and by having a certificate structure that supports access control decisions and more than plain identity certificates.

In SDSI, everything is based on the notion of keys as principals. A SDSI principal is a public key used as a digital signature verification key. There is no notion of "users". Instead, a key can act as a proxy for a user. Thus, in SDSI, a principal is a public key, and is the "one who speaks". If two principals are the same public key, they are taken as being the same.

SDSI does not require a global authentication hierarchy. Instead each principal is a certification authority and can make signed statements. A principal can choose any policy and procedure it likes when signing statements.

Names [Needham, 1989] are central to the operation of SDSI, and each principal can build its own local name space. However, in an operational environment, typically one or more distinguished naming hierarchy roots will also exist (e.g., the Domain Name Service (DNS), and X.509). Global names can be bound to local names within each principal's local name space, and thereby provide means to authenticate principals in other domains.

In SDSI, each signed statement a principal makes, is issued in the form of a certificate. The certificate consists of human-readable contents expressed in free-form ASCII text. Lisp-like S-expressions are used as the structuring tool to express the semantics of the certificate. This approach is more flexible than the well-known identity certificates (e.g., as described in recommendation X.509 [CCITT, 1991]). Identity certificates are used to bind the names of individuals to their public keys. In SDSI, a certificate may contain arbitrary S-expressions and nested levels of signed statements including delegations and access policies.

6.3. Architectural overview

This section describes the μ-CAP authorisation and access control architecture in more detail.

μ-CAP supports the version control access model of FR by providing the following two access request services:

Non-propagated access: Non-propagated (plain) access requests are used by file owners and involve direct interaction between the owner and the server that maintains the file in question.

Propagated access: Propagated access requests enable file owners to hand off some access rights to other users and thereby enabling them to access the files in question.

The access request types are illustrated in more detail in Figure 6.1. In this Figure, A represents the file owner, B the file borrower, and S the file server. Non-propagated access is achieved using a request-response protocol between the owner and the server. Propagated access is achieved using a request-response protocol between B and S where the request contains an explicit hand-off of authority from A to B. Thus, a handoff must be received by B from A prior to any access requests are sent to S by B.

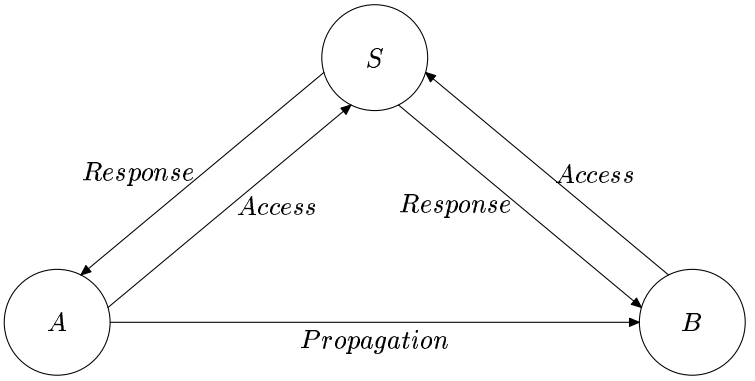

FIGURE 6.1. Access request patterns in μ-CAP

Initial access rights to create objects[3] are granted by the object server, and must have been obtained prior to plain access requests. A user sends a message to the object server requesting an object of a specific type (e.g., a file) to be created. If the user is authorised to create new objects on this particular object server, the object server creates the new object and returns an object identifier to the user.

In μ-CAP, hand-off of authority (e.g., to access a file on a remote server) is based on the use of delegation certificates. A delegation certificate is an expression signed by the object owner stating and authorising the requested operation. The delegation certificate represents a capability that can be used once-only by whomever it has been delegated to. It contains a human readable expression that the object server is able to interpret. The general structure of a delegation certificate representing a signed statement is the following:

$$[\textit{signature}\ \texttt{authorises}\ \textit{action}] \tag{6.1}$$

The *signature* is created by the authorising principal and the *action* represents the access rights granted by the delegation certificate. The term *action* signifies that access rights can be compound and expressed in a programming language, e.g., in PolicyMaker or SDSI notation. The term *capability* is appropriate to describe the delegation certificate construct.

The owner of an object can sign a delegation certificate that, when used by another principal, enables this principal to access the object. In the delegation certificate the action describes the access policy which is being propagated and the signature vouches for its validity. A timestamp is also embedded into the certificate and is used to limit the lifetime in which it should be considered valid. The object owner can define arbitrary access policies and propagate these rights

[3]In the rest of this chapter, the more generic term *object* is used instead of *file* to denote resources subject to access restrictions. Similarly, the term *object server* is used to denote a system maintains objects.

to other principals. The validity of a delegation certificate and its included access rights are verified by the object server upon reception of an access request.

Access control is performed by the object server whenever a request to access a file is received. The object owner needs to trust the object server to carry out authorisation and access control faithfully and according to the policy dictated by the delegation certificate.

Since the object owner may not always be in a position to communicate directly with the object server, revocation of access rights is a problem. Consider a situation where the owner has delegated some access rights for an object to another user, but is prevented from communicating with the object server directly. The user that the object owner has delegated access rights to, may still be in a position to exercise these delegated access rights. As far as the object server is concerned, a user who presents a valid delegation certificate that has been issued by the object owner, is entitled to access the object according to the policy expressed by that certificate. The solution exploited in μ-CAP is to make the delegation certificates valid for one-time usage. Whenever a delegation certificate is presented to an object server, its rights are automatically revoked by the server.

For performance reasons it is important that the revocation is implemented efficiently and that revoked certificates can be processed using minimal resources on the server. An enforcement of "once only" semantics is primarily a question of modifications to the access control mechanism implemented by each of the object servers. It is sufficient to keep a revocation list of all previously invoked requests and check against this list every time a new request occurs. A requirement is that that delegation certificates can be distinguished from each other.

Authentication of users and services is achieved using public-key cryptography. A public-key pair is associated with each user and service, and identity certificates are used to bind user and service identities to public keys.

If confidentiality is also required, it can be built into the propagation and access protocols of μ-CAP, or rely on a network-level security subsystem (e.g., SCIP, see Chapter 7).

6.3.1. Assumptions. This section states the assumptions being made for the μ-CAP architecture in more detail.

- The main assumption underlying μ-CAP is concerned with the confidentiality of private keys and integrity of public keys. Public keys are used for authentication purposes and for verification of delegation certificates created by holders of private keys. They are also used to establish secure communication channels with other principals.
- Honest principals authenticate access requests properly. Honest behaviour is fundamental to the correct operation of any access control scheme.
- A user trusts an object server to perform proper authentication and authorisation of all requests to access objects. Consider the situation when a user does not trust an object server to carry out proper authentication and access control. That is, if users lack confidence in that an object server is capable of storing and protecting objects belonging to them, they

should not rely on this object server in the first place. Thus, the assumption about trust is simply that users express their trust in the correct operation of an object server by storing objects on it.

- A delegation certificate is unforgeable (for a more elaborate description of the underlying assumptions, see Chapter 3). Any attempt to forge a delegation certificate can be detected.

  Another related assumption is that delegation certificates contain sufficient entropy not to be easily forged. There are two issues involved:
  (1) since delegated authority can be exercised only once, each delegation certificate must be unique, and
  (2) the probability of a forged certificate to be taken as authentic by the server (for example, by a birthday attack) is minimal.
- Object servers authenticate the access policies embedded in the delegation certificates. An access policy can be relatively simple, or a complex statement written in a programming language. The requirement is that the object server can execute validated statements. In this respect the assumptions are the same as those underlying PolicyMaker. It is important that each delegation certificate can be interpreted unambiguously and therefore that there is just one interpretation of a statement. The use of a formal language can be used to decrease the probability of ambiguity. It is also assumed that the language used, does not have undesirable side effects such as properties that can compromise the rest of the system.

6.3.2. Non-propagated access protocol. Object owners invoke operations on their objects using non-propagated (*plain*) access requests. Non-propagated access requests are implemented by a two-message Remote Procedure Call (RPC) [Birrel and Nelson, 1984] protocol exchange between the object owner and the server maintaining the object in question. In the first protocol message, the object owner A presents an access request to the server S, and in the second message, S returns the response to that access request back to A. The protocol is modelled as a fail-stop protocol (for a description of the properties of fail-stop protocols, see Section 3.7.3).

More specifically, the non-propagated access protocol consists of the following two messages:

$$\begin{array}{lll} \text{Message 1} & A \rightarrow S: & \{A, S, N_a, C_a\}_{K_s} \\ \text{Message 2} & S \rightarrow A: & \{\{S, A, N_a, F(C_a)\}_{K_s^{-1}}\}_{K_a} \end{array}$$

In the protocol description, A denotes the object owner, S the object server, N_a a nonce invented by A and used as freshness identifier, C_a the delegation certificate created and signed by A, and $F(C_a)$ the response from S to A's access request. $F(C_a)$ represents the response generated by the server of accessing OB, e.g., when a file is read and returned to the A.

The delegation certificate C_a used in Message 1 is encoded as follows:

$$\{A, S, T_a, OB, AC_a\}_{K_a^{-1}} \tag{6.2}$$

A and S denote the object owner and server respectively, T_a a timestamp generated by A, OB the object identifier, AC_a the access rights of A delegated OB. The delegation certificate is self-authorising in that it states explicitly what access rights A should have, and vouches for it using A's signature.

6.3.3. Propagated access protocol. Using a propagated access protocol, object owners can delegate once-only access rights for their objects to other principals enabling them to access these objects. The protocol is similar in structure to the non-propagated access protocol in that it is a request-response communication exchange with the object server. The main difference lies in that the access request is not invoked by the user owning the object. Instead, a delegation certificate containing the delegated access rights is propagated to the borrower prior to the access request.

More specifically, the propagated access protocol consists of the following three messages:

$$\begin{array}{lll} \text{Message 1} & A \rightarrow B: & \{A, B, C_a\}_{K_b} \\ \text{Message 2} & B \rightarrow S: & \{\{B, S, N_b, C_a\}_{K_b^{-1}}\}_{K_s} \\ \text{Message 3} & S \rightarrow B: & \{\{S, B, N_b, F(C_a)\}_{K_s^{-1}}\}_{K_b} \end{array}$$

In the protocol description, A denotes the object owner, B the borrower, S the object server, C_a the delegation certificate created and signed by A for B, and $F(C_a)$ the response from S to B's access request, and N_b a nonce invented by B. $F(C_a)$ represents the response generated by the server of accessing OB.

The delegation certificate C_a used in Message 1 and Message 2 is encoded as follows:

$$\{A, B, S, T_a, OB, AC_b\}_{K_a^{-1}} \tag{6.3}$$

A, B and S denote the object owner, borrower and server respectively, T_a a timestamp generated by A, OB the object identifier of the object to be accessed, AC_b the access rights of B for OB.

Note that propagated access rights can be anonymous. Anonymous delegation of access rights is achieved by not mentioning (or encrypting for) B in any of the protocol messages. A μ-CAP delegation certificate can thus be propagated to any principal and still be enforced by S. If A requires explicit authentication of B, the propagated access rights are delegated to B and stated in the delegation certificate. S authenticates that the request is indeed invoked by B before access is granted. In the case of anonymous requests, B's signature in Message 2 is not required. Similarly, in Message 1, A's encryption of the delegation certificate with B's public key is also optional.

6.3.4. Revocation. Recall from Section 6.3 that μ-CAP's design dictates that object owner are responsible for defining access policies for the objects they own, and to create delegation of access rights expressing these rights. The servers, on the other hand, are responsible for the enforcement of the access rights and the

revocation of them after use. Each object server implements a protection mechanism which is used to enforce the access policies defined by the issued delegation certificates.

The once-only property of μ-CAP's delegation certificates poses some implications for the implementation of the protection mechanism implemented in the object servers. In particular, each object server must revoke the access rights expressed in a delegation certificate immediately after the certificate has been used to grant access, and thereby guaranteeing that each certificate is valid only once.

Several approaches can be used to enforce the once only semantics of μ-CAP's delegation certificates. Solutions typically rely on the use of one or a combination of several of the following approaches:

Revocation lists: Similar to revocation lists in capability-based systems, lists of delegation certificates that have already been used to grant access can be used to identify delegation certificates as duplicates,

Timestamps: synchronized clocks and a time frame in which a delegation certificate is considered valid, or

Message uniqueness: the uniqueness of each delegation certificate is used to determine whether a particular certificate has already been used.

In μ-CAP, revocation of issued delegation certificates and enforcement of once-only access rights relies on the use of a combination of the techniques mentioned above. In the following the revocation procedure is described in more detail.

Revocation lists are used to implement revocation of access rights. A list of delegation certificates that have already been used is associated with each object and maintained by the object server. Upon each request to access an object the revocation list is consulted before an access is granted. The access rights of a delegation certificate are immediately revoked and the revocation list updated to reflect the revoked rights.

Each delegation certificate issued is unique. Uniqueness is guaranteed by numbering and the included timestamp provided by the user. The server uses uniqueness information when updating the revocation list of an object to determine whether the same delegation certificate has already been used.

To simplify the revocation process, each delegation certificate is issued within a specific *epoch*. An epoch is simply an agreement between the user and the server on the maximum number of delegation certificates that at any time can be issued by the user. The main purpose of an epoch is to put an upper limit on the length of the revocation list (and storage space) that the server maintains for each object.

Delegation certificates issued in the same epoch are not ordered and can be presented to the object server in arbitrary order. In this respect, the semantics of μ-CAP's delegation certificates differs from one-time password in other proposed solutions (for example, [Lamport, 1981] and S/Key [Haller, 1994]), where all passwords must be presented in the correct order.

Timestamps are used as an additional source of information for revocation purposes. Since individual users specify access policies, the correctness of the time stamp encoded into each delegation certificate depends entirely on this user's ability

to determine what the current time is. However, in μ-CAP, timestamps are only used to catch up with old certificates.

A user has the opportunity to revoke an entire epoch including all the delegation certificates issued in it. Epoch revocation, however, requires online access to the server. Revocation of an epoch in μ-CAPcan be compared to issuing a new internal capability for an object in ICAP. In both systems the effect is to discard the current revocation list without compromising the security.

6.4. Protocol analysis

This section analyses the non-propagated and propagated access protocols employed in μ-CAP. Both protocols are analysed using the BAN logic of authentication (see Section 3.7.3). The main goal of the analysis is to show that both protocols achieve authentication whenever this is required, and that there is a timely correspondence between the exchanged messages. The latter mainly proves that the exchanged messages belong to the same execution of the protocol.

Authorisation is not an issue in the BAN analysis of protocols.[4] However, timely reception of authenticated messages containing authorisation statements is fundamental to the correct functioning of the access control mechanism. Enforcement of μ-CAP's once-only semantics of delegation certificates is entirely a question concerning the correctness of the implementation of the access control mechanism in the object servers, and is therefore not analysed further in this section.

The BAN logic's *jurisdiction* construct is used to model the object owner's jurisdiction over access rights and the server's jurisdiction over responses to access request. Usually, in the BAN logic analysis of a protocol, the jurisdiction construct is used to express that a principal has jurisdiction over statements about the quality of keys. In the protocol analysis below, the jurisdiction construct is used to cover beliefs about the quality of access requests and responses.

The fail-stop (or rather fail-safe) properties of the μ-CAP protocols are not analysed. However, it can be shown that both protocols satisfy the requirements for such protocols, as outlined in Section 3.7.3.

6.4.1. Non-propagated access protocol. The first step of the protocol analysis is to transform the protocol into an idealized form that can be analysed with the rules defined in the BAN logic. In the idealized protocol, the parts of messages that are sent in clear text or that do not contribute to the beliefs of the recipient are omitted. The idealized non-propagated access protocol becomes:

$$\begin{array}{lll} \text{Message 1} & A \rightarrow S: & \{N_a, \{T_a, X_a\}_{K_a^{-1}} \textit{ from } A\}_{K_s} \\ \text{Message 2} & S \rightarrow A: & \{\{N_a, X_s\}_{K_s^{-1}}\}_{K_a} \end{array}$$

The delegation certificate is in the idealized form of the protocol transformed into

$$\{T_a, X_a\}_{K_a^{-1}} \textit{ from } A$$

[4]Authorisation is also beyond the scope of what the BAN logic is designed to achieve.

In the delegation certificate, X_a represents A's access request. Similarly, in the response from S, X_s represents the result of the access request.

In the protocol analysis, the following assumptions concerning the initial beliefs of principals are made:

$$\begin{array}{ll} A \mid\equiv \overset{K_a}{\mapsto} A & S \mid\equiv \overset{K_s}{\mapsto} S \\ A \mid\equiv \overset{K_s}{\mapsto} S & S \mid\equiv \overset{K_a}{\mapsto} A \\ A \mid\equiv \sharp(N_a) & S \mid\equiv \sharp(T_a) \\ A \mid\equiv S \mid\Rightarrow X_s & S \mid\equiv A \mid\Rightarrow X_a \end{array}$$

It is assumed that both A and S know their own secret key and the other's public key. In addition, A believes that N_a is fresh and S believes that A's timestamp T_a is fresh. S believes A has jurisdiction over the access request X_a, and A believes S has jurisdiction over the response to the access request X_s.

The protocol analysed. S receives Message 1, and from the rules of *message meaning* and *nonce verification* (treating T_a as a nonce), obtains:[5]

$$S \mid\equiv A \mid\equiv X_a \tag{6.4}$$

The *jurisdiction rule* enables S to obtain:

$$S \mid\equiv X_a \tag{6.5}$$

which is a necessary requirement for S to fulfill the access request.

Similar to Message 1, A receives Message 2, and from the rules of *message meaning* and *nonce verification*, obtains:

$$A \mid\equiv S \mid\equiv X_s \tag{6.6}$$

Finally, the *jurisdiction rule* enables A to obtain:

$$A \mid\equiv X_s \tag{6.7}$$

which implies that A believes X_s to be the proper response to the access request.

At the end of the protocol execution S believes it has served an access request that A uttered recently. In addition, A believes the X_s is a timely response to the access request.

6.4.2. Propagated access protocol. The messages of the propagated access protocol are transformed into idealized form using the same approach as for the non-propagated access protocol. The idealized propagated access protocol becomes:

$$\begin{array}{lll} \text{Message 1} & A \to B: & \{\{T_a, X_b\}_{K_a^{-1}} \textit{ from } A\}_{K_b} \\ \text{Message 2} & B \to S: & \{\{N_b, \{T_a, X_b\}_{K_a^{-1}} \textit{ from } A\}_{K_b^{-1}}\}_{K_s} \\ \text{Message 3} & S \to B: & \{\{N_b, X_s\}_{K_s^{-1}}\}_{K_b} \end{array}$$

In the idealized protocol, the delegation certificate contains X_b, which represents B's delegated access rights to A's object OB.

In the protocol analysis the following assumptions concerning the initial beliefs of principals are made:

[5]Intermediate steps of the protocol analysis are omitted from the text.

$$
\begin{array}{ll}
A \mid\equiv \overset{K_a}{\mapsto} A & S \mid\equiv \overset{K_s}{\mapsto} S \\
A \mid\equiv \overset{K_s}{\mapsto} S & S \mid\equiv \overset{K_a}{\mapsto} A \\
A \mid\equiv \overset{K_b}{\mapsto} B & S \mid\equiv \overset{K_b}{\mapsto} B \\
B \mid\equiv \overset{K_b}{\mapsto} B & B \mid\equiv \overset{K_a}{\mapsto} A \\
B \mid\equiv \overset{K_S}{\mapsto} S & \\
A \mid\equiv S \mid\Rightarrow X_s & S \mid\equiv A \mid\Rightarrow X_b \\
B \mid\equiv S \mid\Rightarrow X_s & B \mid\equiv A \mid\Rightarrow X_b \\
B \mid\equiv \sharp(N_b) & \\
B \mid\equiv \sharp(T_a) & S \mid\equiv \sharp(T_a)
\end{array}
$$

It is assumed that A, B and S know their own secret key and the other's public key. In addition, B believes that N_b is fresh and S believes that A's timestamp T_a is fresh. S believes A has jurisdiction over the access request X_b, and A believes S has jurisdiction over the response to the access request X_s. B also believes A has jurisdiction X_b, and that S has jurisdiction over X_s.

The protocol analysed. B receives Message 1 including the delegation certificate from A, and from the rules of *message meaning* and *nonce verification* (treating T_a as a nonce), obtains:

$$B \mid\equiv A \mid\equiv X_b \tag{6.8}$$

The *jurisdiction rule* enables B to obtain:

$$A \mid\equiv X_b \tag{6.9}$$

which indicates that B believes A recently uttered X_b.

S receives Message 2 from B including A's delegation certificate relayed by B, and from the rules of *message meaning* and *nonce verification* (treating T_a as a nonce), obtains:

$$S \mid\equiv A \mid\equiv X_b \tag{6.10}$$

The *jurisdiction rule* enables B to obtain:

$$S \mid\equiv X_b \tag{6.11}$$

which is a necessary requirement for S to fulfill the access request.

B receives Message 3 from S, and from the rules of *message meaning* and *nonce verification*, obtains:

$$B \mid\equiv S \mid\equiv X_s \tag{6.12}$$

Finally, the *jurisdiction rule* enables B to obtain:

$$B \mid\equiv X_s \tag{6.13}$$

which implies that B believes X_s to be the proper response to the access request.

At the end of the protocol execution S believes it has served an access request that B uttered recently containing access rights delegated by A. In addition, B believes the X_s is a timely response to the access request.

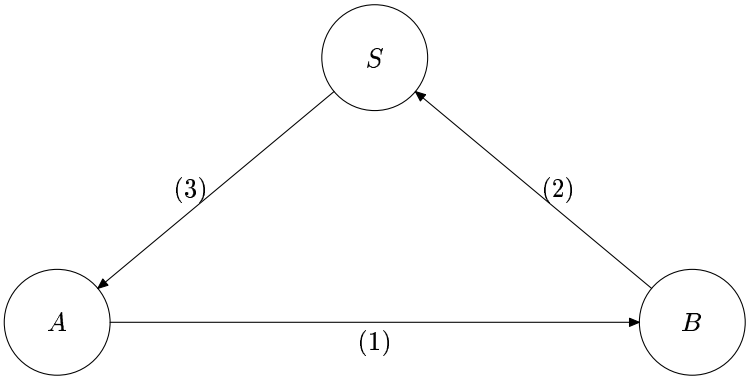

FIGURE 6.2. Conceptual model of electronic payment systems

6.5. Electronic payments

μ-CAP can also be used as the underlying mechanism of a simple electronic payment system. The central idea is to let μ-CAP's delegation certificates represent either electronic cash issued by a bank, or electronic cheques signed by customers of the bank:

Cheque model: In the cheque model, μ-CAP's delegation certificates are used as authorisation statements for actions to take place at the bank (for example, to deduct money from an account.

Cash model: In the cash model, the chain of authority starts at the bank. Similar to μ-CAP's epochs for delegation certificates, the bank issues a series of signed statements with once-only access and propagates these to the user. The user, in turn, can propagate these at will to other users who can return them to the bank.

These two approaches are illustrated in Figure 6.2. In the case of digital cheques, the delegation certificate issued by A allows B to withdraw money from A's account maintained by the bank S. In the case of digital cash, S issues delegation certificates to A, and A forwards these to B as payment tokens. Both scenarios rely on that S is capable of enforcing the once-only semantics. However, since S represents the bank, it is likely in it's own interest to actually do so.

6.6. Discussion

The μ-CAP architecture is to large extents inspired by the ICAP architecture. However, since the underlying assumptions about the operational environment are different in these two systems, the solutions are also inherently different. In particular, in ICAP the responsibility of creating capabilities is assigned to the server which hands these over to clients via secure and trusted channels. In μ-CAP, the same responsibility is assigned to the clients.

A consequence of this design choice was to enforce once-only semantics on the issued capabilities. The reason is that in a computing environment where access control cannot entirely rely on the presence of online services and a fully connected network, traditional capabilities are not well suited. The automatic revocation enforced by μ-CAP solves this by providing a method of revocation that works also for environments in which network links might fail or some machines are off-line.

μ-CAP's protection mechanism depends critically on the correctness of the revocation mechanism and its ability to identify delegation certificates that have already been used.

6.7. Summary

This chapter described the design of an authorisation and access control system based on the use of capabilities with extended semantics. In the proposed design capabilities are implemented as delegation certificates with once-only semantics carrying identities and timestamps. Each capability is automatically revoked by a server when used to access the associated object. Used capabilities are tracked by the servers, and checked against when new capabilities are used. These capabilities can be used to implement authorisation and access control in electronic-commerce applications, and to implement simple payment systems.

—7—

Network-level Security in SCIP

This chapter describes the design and implementation of SCIP, a network-layer authentication and key negotiation system for Internet-like environments. SCIP's system services include authentication, key management and confidentiality. SCIP is implemented as a new communication protocol to be used in conjunction with TCP/IP networks, and it is based on the use of data encapsulation and virtual network interfaces.

SCIP uses an off-line authority to sign public-key certificates for all the machines with which secure communication is required. Certification authorities can be maintained by individual users or by smaller organisations in trusted environments.

The main system service of SCIP is to provide secure communication channels between pairs of mutually authenticated machines at the network-level. Applications can take advantage of the new security services provided by SCIP without having to be modified or re-implemented.

SCIP is designed for small personal computing environments in which individual users manage their own machines and use them to access remote services. The system is designed such that it can be easily extended and used by a larger user community in wide area networks. Most components are implemented in user-space entirely, and the current implementation is small in size, and requires only minor changes to the host operating system. The SCIP prototype is hosted on the NetBSD, a Unix-like operating system that is derived on the BSD-4.4Lite distribution of software.

This chapter is structured as follows. First, an overview of the problem is presented together with a short review of related work. Following the introduction is a description of the SCIP architecture and the requirements underlying its design. The presentation covers data flow in SCIP and the state machine of SCIP's authentication and key negotiation protocol. A section is devoted to a detailed study of the protocol implemented in SCIP together with an analysis of its properties. The chapter finishes with a discussion of the design and a summary of findings.

7.1. Introduction

A problem in contemporary distributed computing environments is the lack of widespread use of network security services and the increasing availability of public resources. Reports of computer fraud and hacker activity in computer networks and

the commercial exploitation of public networks indicate that a high level network security is urgently required.

In particular, a major problem with the current Internet Protocol (IP) [Postel, 1981a], is the lack of support for security functions. In addition, no infrastructure for authentication, integrity and confidentiality of communicated information currently exists or is in widespread use.

Many operating systems hosting IP, also host applications whose security relies entirely on the authenticity and integrity of the network information received from IP or application protocols. In particular, these applications rely on Internet domain names being mapped to correct IP addresses, and that forged communication is detected and discarded by the system. Current TCP/IP implementations do not meet these requirements. This is highly undesirable.

In some systems (e.g., Kerberos, see Section 3.8), security functions have been added at the application level, and are only accessible to modified applications. It is problematic that to take advantage of new security functions, the applications have to be be rewritten. When security functions are added to the network level, changes to applications can be kept to a minimum and will in some cases not required at all.

7.2. Related work

This section describes a set of systems with functionality similar to that in SCIP. The swIPe [Ioannidis and Blaze, 1993] system was one of the first network-level authentication systems proposed. SKIP [Caronni et al., 1996] is implemented by SUN Microsystems. IPv6 [Hinden, 1996] is specification of the new IP standard, and it includes support for authentication, integrity and confidentiality.

SwIPe is a network layer security system that provides data confidentiality and integrity, and source network address authentication for communication between Unix systems. Is is designed to be incorporated into an existing IP implementation without altering the structure of IP itself. This is achieved by tunnelling ordinary IP packets inside IP packets. These packets carry a unique swIPe IP protocol number. SwIPe intercepts and processes these packets separately before they are passed on either upwards or downwards through the IP protocol stack. This approach provides security mechanisms at a fairly low level of the operating system.

An advantage of this approach is that tunnelled packets can transit parts of the network that do not know about swIPe, and there is no need to change existing network infrastructures and interfaces. SwIPe can be used to implement both end-to-end and Network-level security.

The swIPe architecture consists of three conceptual components; the security processing engine, the key management engine, and the policy engine. The policy engine examines outgoing packets to determine whether they should be processed by swIPe, and it examines incoming packets to decide whether they should be accepted and the kind of processing that should be applied. The key management engine establishes session keys and communicates with key management engines on other hosts in order to achieve this. The security processing engine is added to

the network layer to apply the actual security processing to incoming and outgoing packets. Both the key management engine and the policy engine are implemented as user-space processes while the security processing engine resides inside the Unix kernel.

In swIPe, keys and access policies are maintained on a host-by-host basis. Key exchange is done out-of-band and the IP packets carrying these requests are not processed by swIPe itself. The access policy for packets is either *accept* or *deny*, and each host keeps a table containing hosts it accepts swIPe packets from.

SKIP (Simple Key-Management for Internet Protocols) is an implementation of a network layer key management system for IP datagrams, implemented by Sun Microsystems and incorporated into the Solaris Unix operating system. SKIP utilizes Diffie-Hellman public key cryptography [Diffie and Hellmann, 1976] to establish shared secrets that will be used as session keys.

The main components of SKIP are: a key manager daemon which maintains a database of certificate information, a bulk data crypt engine that provides encryption and decryption services to applications, and a TCP/IP streams module which is inserted between the Solaris TCP/IP stack and the network interface.

Recent standardisation efforts have resulted in a new version of the TCP/IP protocol suite called IP Next Generation, protocol version 6 (IPv6). IPv6 has been recommended to be the new Internet protocol by the Internet Engineering Task Force (IETF) and documented in RFC 1752 [Bradner and Mankin, 1995], and approved by the Internet Engineering Steering group as the new IP standard that will eventually replace the current version (IPv4) of the protocol.

The security extensions of IPv6 are two-fold, and the features are documented in RFC 1825 [Atkinson, 1995c]. The primary mechanism is an IPv6 Authentication Header [Atkinson, 1995a]. The authentication header is an extension header that provides authentication and integrity of IPv6 datagrams. The extension header format is algorithm independent and supports different authentication techniques. The secondary extension is the IPv6 Encapsulation Security Header [Atkinson, 1995b]. The security header provides integrity and confidentiality to IPv6 datagrams. Is is also designed to be algorithm-independent. DES in CBC mode (see Section 2.2.1) has been adopted as the standard algorithm to use for data encryption.

7.3. The SCIP architecture

SCIP is a network firewall architecture for personal computers, with a secure protocol for communication with other SCIP hosts through the firewall. A description of SCIP's initial design can be found in [Helme and Plaggenmarsch, 1995].

SCIP's design is influenced swIPe and shares many attributes with the latter. In particular, both SCIP and swIPe use virtual networks and IP encapsulation [Bellovin, 1990] to coexist with other protocols. SCIP also provides authentication, key distribution, and firewall filtering of IP datagrams.

In the following, the SCIP architecture is described in more detail. In particular, SCIP's operational modes, the requirements underlying operational use,

SCIP's protocol formats, access policies, and authentication and key distribution are presented.

7.3.1. Operational modes. SCIP can be configured to be used in three different operational modes, and tailored to fit target environments with different requirements to access policies. The main modes of operation are: host-host communication, remote site communication, and the use of SCIP within conjunction with IP firewalls:

Host-host communication: SCIP is designed to provide secure communication between pairs of hosts, and the host-host communication operational mode is just this. Both hosts authenticate each other mutually using their public keys and negotiate a session key that is used to encrypt subsequent communication between them.

Remote site communication: SCIP can be used to forward IP traffic destined to a particular host to a remote site. In order to use this facility, the host in question must have an temporarily assigned IP address at the remote site that SCIP traffic is forwarded to. The remote site communication facility provides secure network access for machines that are used from insecure domains.

Firewall mode: SCIP can be used as an IP firewall and configured to discard all IP data traffic except that generated by SCIP itself. The firewall mode is used to allow authenticated traffic through the firewall while at the same time rejecting all other communication.

7.3.2. Requirements. The design goal of SCIP is to provide a clean implementation of secure IP that does not alter existing TCP/IP protocol implementations or operating systems substantially. In this process, the following requirements are identified:

Flexibility: On each host that supports SCIP it should be easy to add new access policies or make changes to existing policies governing access to that host. Each host should be able to define and enforce its own policies independent of the policies defined and enforced by other hosts. In addition, inbound and outbound data traffic should be treated separately in such way that the policy for inbound traffic can be different from the policy for outbound traffic.

Modularity: The system should be modular and extensible. Adding new formats and/or protocols and algorithms should not require a rewrite of the existing system.

Compatibility: The system should be compatible with existing network protocols and easy to incorporate into an existing network infrastructure. That is, the extensions that SCIP makes to TCP/IP should not alter the existing protocol formats in any way that renders it incompatible with IP.

Efficiency: The implementation should be fast enough for ordinary workstation workload on local area networks, including remote login sessions,

WWW access, and remote file system operations. The impacts of high-speed networks and multimedia applications are not considered. Mechanisms should be implemented in the Unix kernel.

Pairwise communication: Authentication and the establishment of secure channels has to be between pairs of hosts. This enables each host to specify access policies with a reasonably high level of granularity.

Stateless operation: Protocol units should be protected independently from each other. The loss of one datagram should not affect the processing of any other datagram. Since SCIP is applied at the network level, all higher level transport protocols will automatically retransmit lost datagrams. In this respect, SCIP does not differ from standard IP.

Forward secrecy: If a host is compromised, forward secrecy should still be provided. In particular, if a long-term private key of a host is compromised, it should not be possible to decrypt past communication between that host and other hosts.

7.3.3. Protocol format. In the following, a more detailed description of SCIP's protocol formats are presented.

Hosts that support SCIP exchange data using a dedicated datagram format. The format used by SCIP is interchangeable with existing IP datagram formats. In order for the TCP/IP protocol to distinguish between SCIP datagrams and normal IP datagrams, an unassigned protocol number is used as the SCIP protocol identifier. The protocol number used by SCIP is `IPPROTO_SCIP` = 133 (an overview over other assigned protocol numbers can be found in [Reynolds and Postel, 1994]). The SCIP protocol header formats are similar to the IPv6 security header extensions, but not compatible with them.[1]

In order to to process a datagram at the receiving side, a subheader containing information about the protection scheme in use, together with administrative information (e.g., the digital signature scheme being used) is added to the payload of the datagram. The source and destination addresses in the header of the encapsulated datagram are typically the same as those of the original datagram being encapsulated.

The format of the SCIP protocol units exchanged is illustrated in Figure 7.1 on page 98. A standard IP datagram is the payload of the SCIP datagram. The payload of the SCIP datagram can be (optionally) encrypted (not shown in the figure). The header contains the same information as the original datagram except for the protocol identifier which is set to `IPPROTO_SCIP`. SCIP datagram options are used to describe the method that has been used to protect the datagram, e.g., the encryption algorithm identifiers and digital signatures.

The source and destination header fields of a SCIP datagram can be different from those of the encapsulated datagram. The purpose of this is to support the use of SCIP in conjunction with IP firewall gateways and forwarding of communication,

[1]At the time when SCIP was designed, no IPv6 secure protocol formats had yet been published.

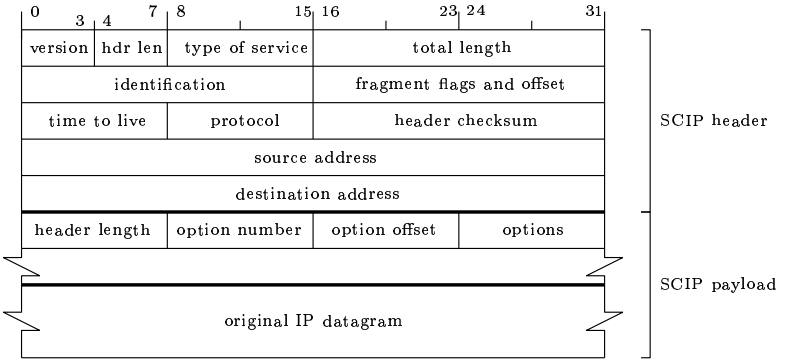

FIGURE 7.1. The SCIP protocol unit

and for experiments with different mobile IP protocols (in particular mIP, see [Voet, 1995, Section 4.2]).

The use of SCIP protocol headers add extra data overhead. The overhead amounts to one IP header plus SCIP options. For normal host-host communication, a header compression scheme (e.g., van Jacobsen header compression for low-bandwidth serial links [Jacobsen, 1991]) can be used to reduce the overhead.

Both the source and the destination addresses of each SCIP datagram are properly authenticated. Authentication is achieved by establishing a session between a pair of hosts, and by providing a shared key that is valid only for communication between these two hosts. Authentication in SCIP is based on public key encryption, and protection of data transfer is based on shared key encryption.

7.3.4. Access policies. In SCIP, access policies are maintained on a per host basis. Each host decides what other hosts it should communicate with using SCIP, and which hosts it should reject communication from. In addition, it is also possible to define policies for specific hosts that a host still can communicate with using conventional IP datagrams. SCIP operation is disabled for all communication with such hosts.

All non-SCIP datagrams are discarded upon reception. This is a matter of policy, and the approach is, in many respects, similar to that of an IP-level firewall where only authenticated datagrams will be mediated through the firewall.

Figure 7.2 on page 99 illustrates the format of a SCIP policy table entry for communication with a host using SCIP. Each policy table contains IP addresses of remote hosts, the public key to use, and time-to-live values for each of the keys. The policy table properly describes the current access policy of the host in question.

7.3.5. Authentication and key distribution. In SCIP, each pairwise connection is protected using symmetric-key encryption. Each key is used for message authentication, integrity and confidentiality, and is negotiated by the two machines involved in a protocol execution. In addition, each key is only valid for a limited period of time. Whenever a key expires, a new key negotiation is initiated.

```
[main]
tunaddr     = 1.1.1.1
scipdebug   = 0
kmdebug     = 0
kerneldebug = 0

cakey       =
   02 03 01 00 01 02 30
   05 53 81 58 2b 77 cc 7a 02 b0 99 bc 4e a4 79 60
   ...

localcertificate =
   00 16 1c 6c
   69 66 6c 61 66 2e 70 65 67 61 73 75 73 2e 65 73
   ...

machinekey =
   b2 dc 1b 57 1c 59 67 e1 1b 24 13 c5 8e 2d 2a 52
   dd 8a dc f7 ad 2b 86 5c 16 e3 54 b0 84 b0 da d3
   ...

[thecity.pegasus.esprit.ec.org]
secure      = FALSE

[damien.pegasus.esprit.ec.org]
secure      = TRUE
active      = TRUE
encryption  = RC5
signature   = MD5
keysize     = 128
lifetime    = 1800
expsize     = 10000000
certificate =
   00 16
   1c 64 61 6d 69 65 6e 2e 70 65 67 61 73 75 73 2e
   ...
```

FIGURE 7.2. Illustration of a SCIP policy table file

The SCIP authentication and key distribution protocol consists of the exchange of the following three messages: `HELLO`, `RESPOND`, and `FINISH`. The first message is sent from a local host A to the remote host B that it should establish a secure communication channel and exchange IP datagrams securely with. Upon reception of the first message, the remote host B generates a random number that it sends

back in the following message. The random number is a session key component. The initiating host also generates it's own random number and sends this to the remote host. After the three messages have been exchanged, both parties possess two random numbers that can be spliced into a new session key. This session key is then used to encrypt all subsequent communication between the two hosts. More specifically, the following exchange of messages takes place:

$$\begin{aligned} A \rightarrow B: &\quad \{A, B, N_a, CK_{n_a}\}_{K_{n_a}^{-1}} \\ B \rightarrow A: &\quad \{\{B, A, N_a, N_b, R_b, CK_{n_b}\}_{K_{n_b}^{-1}}\}_{K_{n_a}} \\ A \rightarrow B: &\quad \{\{A, B, R_a, N_b, \{D\}_{K_{ab}}\}_{K_{n_a}^{-1}}\}_{K_{n_b}} \end{aligned}$$

In the protocol description, $\{X\}_{K_a^{-1}}$ denotes that message contents X is signed with A's secret key,[2] and $\{X\}_{K_b}$ denotes that formula X is encrypted with B's public key.[3] A and B denote the two authenticating parties, N_a is a nonce generated by A, N_b is a nonce generated by B, R_a and R_b are random numbers generated by A and B respectively, CK_{n_a} and CK_{n_b} are the node certificates of A and B. To increase the readability of the protocol description, the identifier and its version number, and the message sequence number are not shown in the protocol description. D is the IP datagram that triggered the execution of the protocol, and it is encrypted with the new session key K_{ab} that has been derived from the session key components R_a and R_b.

The certificate CK_{n_a} expresses the following handoff of authority:

$$K_a \textbf{ says } K_{n_a} \Rightarrow K_a \tag{7.1}$$

and is similar to the handoff of authority as described in [Abadi et al., 1990]. Since it is already assumed that both A and B already know each others public keys (or can learn them via a trusted third party), handoff certificates can also be verified. A trusted third party has signed the long-term public key of each host and published them prior to any communication between the hosts.

The use of node certificates in the protocol description above requires some more explanation. Each machine is entitled to know two public-key pairs. One public key-pair is used for a long term while the other key-pair is only valid for a shorter period of time. The former can be stored on a secure processor and does not need to be online during system operation, while the latter is generated at boot time and is valid for a limited time interval only.

A boot certificate represents the handoff of authority from the long-term key to the short-term key. It can, for example, be signed by a secure processor (e.g., a smart card) at boot time and then made available to the host operating system. In this way some authority is handed over to the host for a limited amount of time.

---

[2]Note that the digital signature operation is applied to $H(X)$, where H is a strong one-way hash function applied to X. Thus, the actual meaning of $\{X\}_{K_a^{-1}}$ is $\langle X, \{H(X)\}_{K_a^{-1}} \rangle$.

[3]For efficiency, formula X is encrypted with a fast data encryption algorithm F and a randomly created key K. The actual encoding is $\langle \{K\}_{K_b}, \{X\}_K \rangle$.

When the system is booted, a public key pair and a session certificate that the host can use to authenticate itself are created. This certificate is handed over to the remote host during authentication and used as part of the authentication process.

Each SCIP host can be booted from a secure processor (e.g., as outlined in Chapter 4). During system initialization and startup a delegation certificate is created by the secure processor and handed over to the operating system together with a secret node key. The delegation certificate is used by a host to prove its authenticity to other hosts, and that the machine has been booted correctly.

It is assumed that a strong trust relationship exists between A and B, reducing the problem to mutual authentication in the presence of an insecure network. Furthermore, it is assumed that both A and B are capable of, and are willing to generate proper random-numbers R_a and R_b to use in the construction of new session keys. A splice function (denoted $\oplus$ in the remainder of the chapter) is used to generate a session key from the two session key components R_a and R_b.

7.4. Implementation

This section describes the implementation of SCIP and how it is incorporated into the NetBSD operating system. An early design decision was to integrate SCIP into a 4.4BSD [McKusick et al., 1996] derived operating system and its implementation of the TCP/IP protocol suite (for a more detailed description of the TCP/IP implementation, see e.g., [Wright and Stevens, 1995]).

In order to avoid problems the fragmentation of IP datagrams, all kernel modifications have been incorporated above the IP fragmentation layer. Due to the flexibility of the 4.4BSD implementation of this protocol suite [McKusick et al., 1996], it is possible to design a system where most tasks are implemented as applications in user-space.

In the following, the main components of SCIP and the data paths between them are described in more detail and some of the more important implementation details are sketched. Included in the description is also a detailed study of the authentication and key negotiation protocol used in SCIP including an analysis of its properties.

7.4.1. Protocol engine. Most of the SCIP protocol engine is implemented as a Unix user space process called *scipd*. The main responsibility of *scipd* is to process datagrams according to the policy tables. That is, outgoing datagrams are encapsulated and incoming datagrams decapsulated.

Using the standard routing mechanisms in NetBSD, outgoing SCIP datagrams are routed by *ip_output*() to the the virtual network driver *sc0*. *scipd* receives these datagrams from *sc0* to perform encapsulation and encryption. The destination address of a datagram is used to find the policy table entry to use for these operations. The session key established with the remote host is used to encrypt the datagram. If no session key is available, or if the existing key has expired, a new key is generated using the key negotiation protocol. Finally, the datagram is encapsulated by creating a new IP datagram from the encrypted datagram, the sub-header and a

new IP header. The encapsulated datagram is then written back to *sc0*, and follows the default IP route to a gateway for the remote host.

Incoming SCIP datagrams are received by *scipd* by means of a raw socket connection with direct access to IP traffic. The source address of each datagram is used as an index into the policy table. In order to decapsulate a datagram to reveal the original datagram, information in the sub-header is checked against that in the policy table to determine what decryption algorithm and digital signature verification scheme to use. If the encryption key needed to decrypt the datagram is expired, the datagram is discarded and negotiation for a new session key is initiated. If the datagram can be decapsulated properly, its payload, the original IP datagram, is written back to *sc0* and then enqueued to the default IP input queue for local processing and delivery by *ip_ipintr*().

Figure 7.3 on page 103 depicts the main system components and the information flow from applications to the network device(s). Information flow is *outbound* for datagrams departing from the host and *inbound* for datagrams arriving at the host. The routing mechanism of the 4.4BSD operating system is used to control and redirect the information flow between the network devices in the system.

Inbound datagrams are processed by a SCIP host as follows. First, a network protocol unit is received by a real network interface (in Figure 7.3 denoted *ed0*), e.g., over an Ethernet. If the unit contains (fragments of) an IP datagram, it is enqueued onto the IP input queue (*a*). In *ip_ipintr*(), a check is carried out by a filter to determine whether the datagram is received via the real network interface and whether it is a SCIP datagram (*b*). If this is the case, the datagram is enqueued to be read by *scipd*, which listens to a raw socket for IP datagrams carrying protocol number `IPPROTO_SCIP` (*c*). *scipd* uses the contents of its policy table to check whether the datagram has been sent from an authorised host that also supports SCIP. If this is the case, and a valid session key for the remote host in question exists, the datagram is decapsulated to reveal the original datagram. Upon successful decapsulation, the original datagram is written to the virtual network device driver *sc0* using a terminal-like interface (*d*). If decapsulation did not succeed a key negotiation request is sent to the sender and the encapsulated datagram is discarded. If the virtual network device driver detects that it is a datagram destined for the local host, it is again enqueued to the IP input queue (*e*). However, since it now originates from *sc0*, it is accepted by the filter and can be processed by higher-level layers of the TCP/IP protocol stack and delivered to applications in the usual way (*f*) and (*g*).

Outbound data follows the data path from applications to the TCP/IP protocol stack in the usual way (*1*), and is processed normally until it reaches the IP layer (*2*). At the IP layer individual datagrams are processed differently according to the contents of the policy table for outgoing datagrams. All SCIP communication with other SCIP hosts is routed via the virtual network *sc0* using the standard kernel mechanisms for routing (*3*), while communication with other hosts are routed to the real network device *ed0* (*7*). *sc0* immediately enqueues a datagram to be processed by *scipd* (*4*), and *scipd* encapsulates the entire datagram in a new datagram with the protocol number set to `IPPROTO_SCIP`, processes the

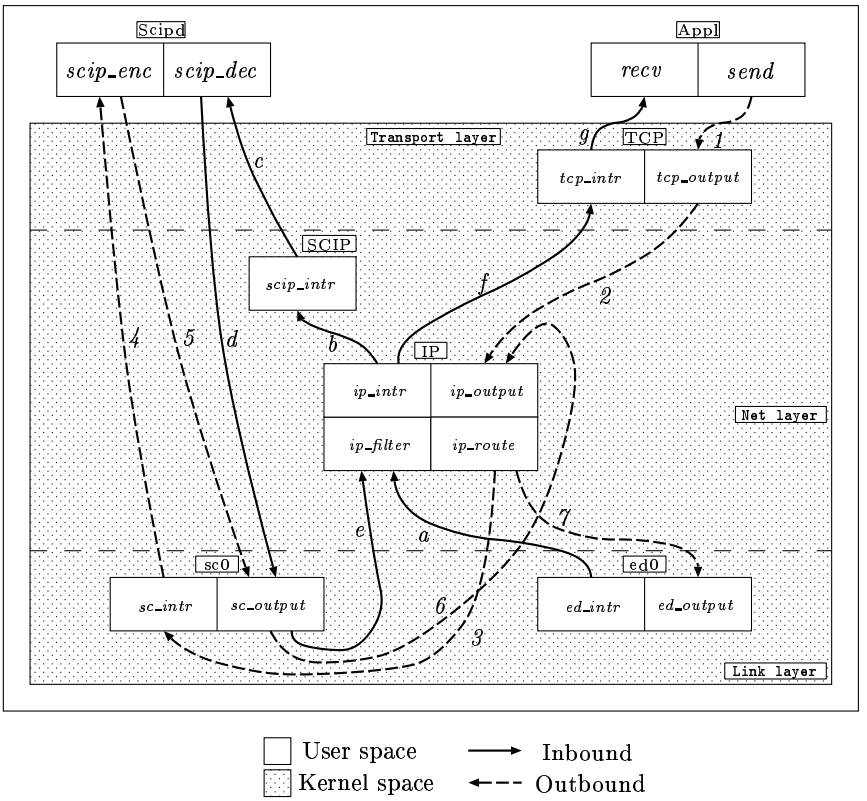

FIGURE 7.3. Inbound and outbound datagram paths

payload according to the policy table entry of the remote host, and writes it back to *sc0* (*5*). This time *sc0* invokes *ip_output*() directly, but this time ignores all routes pointing back to *sc0* (*6*), and the datagram departs from the system via the real network interface (*7*).

7.4.2. Kernel changes. The kernel changes in the implementation are kept to a minimum. This section describes the modifications that were carried out. In particular, the structure of the virtual network device driver and the IP input filters are described.

The virtual network device driver (*sc0*) implements the low level kernel mechanisms of SCIP. One of the functions of this device driver is to make datagrams departing from the TCP/IP protocol stack available to *scipd*. Thus, it acts as if it was a normal network device, but instead of writing a datagram to a hardware device, the datagram is linked to a queue that can be read by a user-space process. The device driver has a standard TTY interface and datagrams are read and written using ordinary Unix file operations.

When a datagram is written to the device driver, the protocol number in the header of the datagram is used to determine whether it should be dispatched for incoming or outgoing processing. If the protocol number is `IPPROTO_SCIP`, the datagram contains only a skeleton IP header as required by *ip_output*(). Some fields, such as source and destination address together with the protocol number are filled in. Before the datagram is handed over to the real network device, the routing table is consulted for a route for a network that can be used to forward the datagram to the remote host. When the routing table is consulted, routes to the virtual network are ignored to prevents loops. If the protocol number is not `IPPROTO_SCIP`, it is a complete IP datagram which is read and enqueued for further processing by *ip_ipintr*(). This option is normally used to re-inject decapsulated datagrams back into the local system for normal IP processing.

The implementation of the input filter is straightforward. It is installed into *ip_ipintr*() and executed after the processing of standard IP options. As a consequence, all incoming datagrams to the local host are checked against the contents of the policy table. The filter always accepts SCIP datagrams. The authenticity of these datagrams is checked by *scipd* later.

The input filter also maintains an exception list containing addresses of hosts from which ordinary datagrams are accepted. This feature, however, should be used with great care, since it degrades the security of the system when being used. Finally, there is another way for non-SCIP datagrams to be accepted by the filter: all datagrams that originate from the local virtual network device or the loop-back interface, are accepted. Since reinjection of datagrams is done by enqueuing them to *ip_ipintr*(), these datagrams are accepted by the filter without any check. If this option is not supplied, decapsulated datagrams are also rejected by the filter.

7.4.3. Authentication and key distribution. The authentication and key distribution protocol used in SCIP and described in Section 7.3.5, is implemented as an event driven state-machine. State information is kept for each remote host that a host communicates with. On each event, an associated function is invoked to perform an event-specific action. The outcome of an action depends on the current local state information and the kind of event that occured. The possible classes of events that can occur are the following:

Request protocol execution: A protocol execution can be requested either by the local host or by a remote host. An execution of the protocol is initiated by the local host if an IP datagram is destined for a particular remote host, and a local policy states that communication with that remote host should be carried out using SCIP. An execution of the protocol is also initiated whenever a host receives a key negotiation request from a remote host.

Incoming protocol message: Incoming protocol messages cause events that may change the state of the protocol execution. Non-SCIP protocols messages, or invalid SCIP protocol messages are discarded. Similarly, invalid SCIP protocol messages are also discarded. A SCIP protocol message is deemed valid and is allowed to change the protocol state if, and

only if, it is expected in the current protocol state, and passes all authentication and integrity checks.

Message timeout: Each protocol-message transmission is guarded by a retransmission timer. In a wait state, a timeout function is used to detect protocol failures in case protocol messages are not received within a specified period of time. Timers are maintained on a per message basis. The value `max_retransmit` determines whether the maximum number of retransmissions has been reached, and causing the protocol to abort.

Each SCIP host maintains a list of SCIP Protocol Control Blocks (SPCB) indexed by IP addresses. For each remote SCIP host with which a protocol execution is initiated, an SPCB is allocated and added to the list. The SPCB is a placeholder for the local protocol state information and holds for each remote host with which communication is initiated the following state information:

- the name and IP address of the remote host in question,
- a nonce created by the protocol initiator,
- random-number session-key components,
- a retransmission counter,
- local state of the protocol execution in question.

In the implementation, hashing techniques are used to speed up access to individual SPCBs.

SPCBs that are associated with remote hosts not communicating with the local host anymore are deallocated. Sensitive information is erased after use, e.g., the implementation deletes all session-key components and session keys that are not in use anymore.

The SCIP protocol state-machine is operated in two different modes; initiator and responder. The following explains the difference between these two modes:

Initiator: The local host is initiating an authentication and key negotiating protocol.

Responder: The local host responds to an authentication and key negotiation request.

The relations between actions, states and events are depicted in Figure 7.4 on page 106. The state transition diagram only shows those events that lead to an action and sometimes a protocol state transition. Invalid messages are simply discarded by the protocol. Retransmission timers and counters are used to prevent deadlocks in the protocol. Each timeout results in a increment of the retransmission counter for the current protocol state, and a transition back to the same protocol state. If the retransmission counter reaches `max_retransmit` from any of these states, the protocol is aborted.

Since either party can take on the rôle as either initiator or responder, both parties execute the same protocol steps where, in a particular protocol execution, one of the parties will be the initiator and the other the responder. The following states can be reached (Figure 7.4):

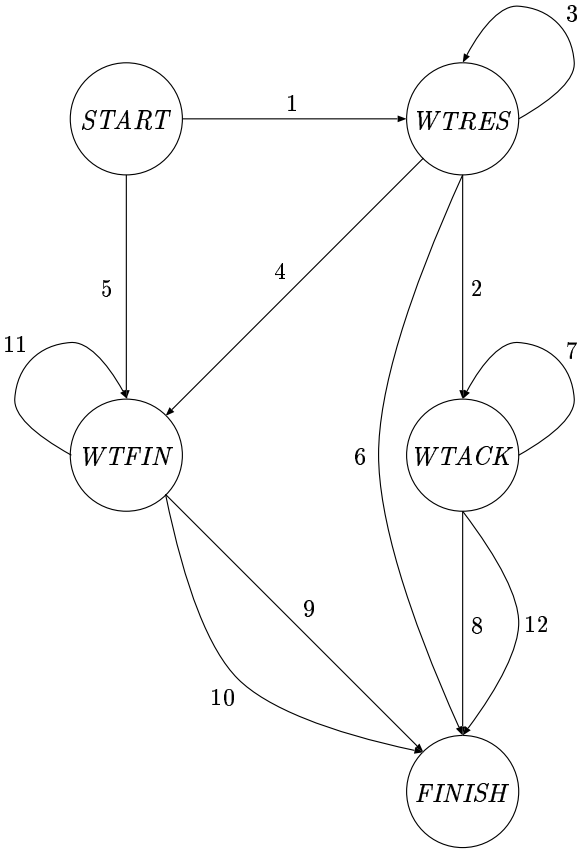

FIGURE 7.4. States and transitions in the SCIP authentication and key negotiation protocol. The circles represent possible states, and the labelled arrows denote state transitions.

START Initial state of the protocol. The host is waiting for either a key negotiation message from a remote host or a local key negotiation start event.

WTRES The host executes the protocol is in initiator mode. A `HELLO` message has been sent. Retransmission counter for `HELLO` messages is running. Initiator is waiting for a `RESPOND` message.

WTFIN Protocol is executing in responder mode. A `RESPOND` message has been sent. Retransmission counter for `RESPOND` messages is running. Responder is waiting for a `FINISH` message.

WTACK Protocol is executing in initiator mode. A `FINISH` message has been sent. A retransmission counter for `FINISH` messages is running. Initiator is waiting for the reception of a datagram encrypted with the new session key.

FINISH Final state of the protocol. The outcome of the protocol execution is either successful, and a new session key has been installed, or unsuccessful and the protocol aborted.

In the following, a more detailed description of the state transitions that can occur are presented. Labels in the description correspond to the labels used in Figure 7.4.

1 Event: *Protocol execution is initiated by the local host.*
Action: A new protocol execution is initiated, an SCPB allocated, and a `HELLO` message is sent to the remote host. Local host protocol mode is set to **initiator**, local state to *WTRES*, and a retransmission timer is started. If a SCPB for the remote host in question already exists, its current contents is deleted to initiate the new protocol execution.

2 Event: *A `RESPOND` message is received.*
Action: If the retransmission timer is running in the *WTRES* state, then it is stopped. A `FINISH` message is sent to the remote host, local state is set to *WTACK*, and a retransmission timer for the `FINISH` message is started.

3 Event: *The retransmission timer expires.*
Action: The retransmission counter is incremented by one, the `HELLO` message is retransmitted, and a new retransmission timer is started.

4 Event: *A `HELLO` message is received.*
Action: If a `HELLO` message arrives while an initiator is in the *WTRES* state, a protocol execution has been initiated by both the local host and the remote host concurrently, and both of them are waiting for a `RESPOND` message. Obviously, one of the two protocol runs has to be aborted, and one of the parties must take on the rôle of the responder. The nonces N_a from the local host and $N_{a'}$ received in the `HELLO` message from the remote host are used to determine whether the local host should become a responder or not.
If $N_a > N_{a'}$ the local host changes from **initiator** to the **responder** mode of operation, any retransmission timer in the *WTRES* state is cancelled, a `RESPOND` message is transmitted, and a state transition to *WTFIN* is made. If $N_a < N_{a'}$ the local host simply discards the currently received `HELLO` message, resets its retransmission timer, and awaits a `RESPOND` message instead.[4]

5 Event: *A `HELLO` message is received.*
Action: A new protocol execution is initiated, a SCPB allocated, and a `RESPOND` message is sent to the remote host. Local-host protocol-mode is set to **responder**, local state to *WTFIN*, and a retransmission timer is started. If a SCPB for the remote host in question already exists, its current contents is deleted to initiate the new protocol execution.

6,10,12 Event: *`max_retransmit` retransmissions have been carried out.*

[4]In the unlikely case that $N_a = N_{a'}$, the IP addresses of both parties are compared instead.

Action: The SPCB and temporary variables (e.g., session key components) are erased and deallocated. The protocol is aborted, and higher layer protocol are notified about the unsuccessful protocol execution.

7 Event: *The retransmission timer expires.*
Action: The retransmission counter is incremented by one, the `FINISH` message is retransmitted, and a new retransmission timer is started.
Note that the retransmission of the `FINISH` message does not necessarily imply that the datagram D is delivered more than once at the destination. The protocol will discard any retransmitted message received after a state transition to *FINISH* has taken place.

8 Event: *A datagram encrypted with the new session key is received.*
Action: The new session key is installed, local state is set to *FINISH*, and higher level protocols are notified about the successful execution of the protocol.

9 Event: *A `FINISH` message is received.*
Action: The new session key is installed, local state is set to *FINISH*, and higher level protocols are notified about the successful execution of the protocol.

11 Event: *The retransmission timer expires.*
Action: The retransmission counter is incremented by one, the `RESPOND` message is retransmitted, and a new retransmission timer is started.

7.4.4. Data encryption. In SCIP, shared-key encryption is used for bulk encryption of data and public-key encryption for authentication. Each IP datagram that is being exchanged between a pair of hosts is encrypted with a session key the two hosts share between them. In SCIP, public-key cryptography is performed using RSA [Rivest et al., 1978], and datagrams are encrypted using RC5 [Rivest, 1995].

Since IP is a lossy protocol, messages do not always reach their destination. This causes few problems for the communication between two SCIP hosts. IP datagrams can be dropped before they arrive at their destination, and when they arrive, they will only be delivered to higher level protocols if they are decrypted successfully. Higher level protocols and correctly implemented applications will retransmit data lost during transmission and ensure connectivity when possible.

A potential weakness is that messages exchanged in SCIP can be vulnerable to known-plaintext attacks. When no special attention is paid to the protection of sequence numbers and other known identifiers inside the encapsulated IP datagrams during implementation, such attacks are more likely to succeed. In SCIP, the contents of each encapsulated IP datagram is encrypted with a stream cipher in cipher block chaining and applied to individual datagrams independently. This makes it harder to launch attacks on sequence numbers and protocol identifiers.

7.4.5. Performance. Reducing the number of RSA operations would improve performance of the protocol significantly. However, in order to not violate the fail-stop property, none of the RSA operations currently used in the protocol can be

removed. Since the run of a key negotiation protocol occurs infrequently, this is not considered to be a major performance problem.

The current implementation of RC5 is implemented in software and optimized for Intel x86-family of processors. On a 66 MHz Intel 486DX2 processor throughput of encrypted datagrams is measured to be 2500 KB/s. On the same architecture, a complete key negotiation takes, typically, in the order of a tenth of a second.

Encapsulation of IP datagrams increases network data overhead. In SCIP, the overhead amounts to an additional IP header per datagram, and an increased and processing load in the protocol stack. Each datagram is extended with one IP header that amounts to 20 additional bytes. In addition, SCIP options in the header may increase the total overhead to 52 bytes in case the IP header is not compressed.

For efficiency reasons, the datagram D that triggered the SCIP protocol authentication and key negotiation protocol is included in the final message of the protocol. When a connection to a new host is established, TCP [Postel, 1981b] sends the remote host a *SYN* message. Fast delivery of the *SYN* message to the remote host yields a short TCP connection setup time. A quick acknowledgement of a *SYN* message also releases the initiator from the "limbo" *WTACK* state described in section 7.4.3. An alternative would have been to simply discard D and let TCP retransmit the *SYN* message when it's timeout expires. The latter alternative, however, would slow down most connection setups significantly.

7.5. Protocol analysis

In this section an analysis of the SCIP authentication and key negotiation protocol is presented. Since the protocol is developed using the fail-stop principle, the proof methodology is also the one of fail-stop protocols (see Section 3.7.3 for more details).

Recall from Section 7.3.5 that the SCIP protocol contains a handoff of authority from the long-term secret key of each host to a node key. The handoff of authority which, is expressed in the first and second message of the protocol, is described by formula (7.1). That is, if B receives the first message and believes that K_a is A's public key, then B can use the handoff axiom P10 from [Lampson et al., 1992a] to infer that the first message was signed by A:

$$\vdash (A \textbf{ says } (B \Rightarrow A)) \supset (B \Rightarrow A) \tag{7.2}$$

In the rest of the analysis, only a simplified expression is used. The analysis is also limited to the use of the secret key of the host only. A similar approach can be used for the second and third messages. Thus, the protocol to analyse is:

$$\begin{array}{lll} \text{Message 1} & A \rightarrow B: & \{A, B, N_a\}_{K_a^{-1}} \\ \text{Message 2} & B \rightarrow A: & \{\{B, A, N_a, N_b, R_b\}_{K_b^{-1}}\}_{K_a} \\ \text{Message 3} & A \rightarrow B: & \{\{A, B, R_a, N_b, \{D\}_{K_{ab}}\}_{K_a^{-1}}\}_{K_b} \end{array}$$

7.5.1. Validating the fail-stop property. The first step of the fail-stop protocol verification process is to check whether the protocol actually is fail-stop.

If the protocol is fail-stop, active attacks do not have to be considered in the remainder of the analysis.

Clearly, the protocol, in its current state, can not be considered fail-stop, but rather fail-safe (for more details, see Section 3.7.3). The reason is that the receiver of the first protocol message cannot verify the freshness of this message. Since, as will be shown later in the analysis, the actual fail-safe property of the protocol has no major security implications, the analysis is conducted as for a protocol that satisfies the fail-safe property.

In the protocol specification, each message contains a header including the names of the sender and the recipient and a message type identifier (not shown above, but present in the implementation). The nonces N_a and N_b are used as freshness identifiers. Therefore, the first requirement of fail-stop (and fail-safe) protocols is satisfied (see Section 3.7.3). Since each protocol message is signed with the sender's private key, the second requirement is also met. The protocol is designed to discard all messages that cause unexpected events and deviations from the expected protocol execution path (see Figure 7.4). As soon as the number of retransmission exceeds the maximum allowed number of retransmissions, the execution of the protocol will be aborted. Thus, it can be concluded that the protocol is fail-safe.

7.5.2. Validating the secrecy assumption. Applying a BAN like logics to any protocol does not make sense unless certain requirements are met. The most important requirement underlying BAN is the secrecy assumption: those data items that are meant to be secret, will not be discovered by an adversary during the protocol execution (discussions of this shortcoming of the BAN logic can be amongst others be found in [Gong et al., 1990, Abadi and Tuttle, 1991, Gong and Syverson, 1995]). In the SCIP protocol, the data items of importance are R_a and R_b, the two random numbers that serve in the construction of a new session key. An adversary might record each message of the protocol and observe:

$$\{\{B, A, N_a, N_b, R_b\}_{K_b^{-1}}\}_{K_a}$$
$$\{\{A, B, R_a, N_b, \{D\}_{K_{ab}}\}_{K_a^{-1}}\}_{K_b}$$

An adversary does, by assumption, not have access to the private key of either A or B and thus, will not be able to remove the outer encryption from the messages mentioned above to reveal the random session key components. From the assumption about the secrecy of the secret keys in use, it can be concluded that the secrecy assumption for the SCIP protocol also holds.

In addition, if one the private keys of either A or B is compromised some time in the future, this will not cause any cascading effects and automatically compromise any of the session keys used between A and B. The reason is that a complete session key is never exchanged encrypted with the private key of either A or B. Thus, both K_a^{-1} and K_b^{-1} must be compromised to recover a complete session key.

This approach requires that both A and B purge all old session key components and session keys when they are not in use anymore. A similar approach to key exchange is is described in [Lampson et al., 1992a].

Note that A may trust B at the time when a key negotiation takes place. However, A may not trust B for all time in the future. If such distrust will occur, A should be rather confident that communication with B that has already taken place in the past cannot be compromised. A good reason for A's new distrust of B might be that of a change of policy, or simply, that someone that A's owner distrusts now works for the organisations responsible for B.

7.5.3. BAN logic analysis. The BAN logic analysis of the protocol follows the standard procedure outlined in Section 3.7.3.

In the first phase of the protocol (`HELLO`, `RESPOND`), N_a is used to convince A that B is participating in the same protocol run, and thus, that the `RESPOND` message is not being replayed. N_b is used in a similar handshake (`RESPOND`, `FINISH`) to convince B, that the remote party A is present and participates in the protocol

The purpose of the `HELLO` message is to transfer the nonce N_a to B.[5] The idealized form of the message consist of the nonce signed by K_a^{-1}.

The `RESPOND` message serves in transferring a nonce N_b generated by B, a random secret number R_b and A's freshness identifier N_a back to A. In the idealized protocol, R_b is written as a secret shared by A and B. Here, The notation serves only for semantic purposes. In practice, only a number is actually being transferred between A and B.

The `FINISH` message contains A's random number R_a and B's nonce N_b. The transfer of R_a is a shared, while N_b is the freshness identifier.

The idealized protocol becomes:

$$\begin{array}{lll}
\text{Message 1} & A \to B: & \{N_a\}_{K_a^{-1}} \\
\text{Message 2} & B \to A: & \{\{N_a, N_b,\ A \overset{R_b}{\rightleftharpoons} B\}_{K_b^{-1}}\}_{K_a} \\
\text{Message 3} & A \to B: & \{\{A \overset{R_a}{\rightleftharpoons} B,\ N_b, \{D\}_{K_{ab}}\}_{K_a^{-1}}\}_{K_b}
\end{array}$$

In the BAN logic analysis of the SCIP protocol, the following initial assumptions are made:

$$\begin{array}{ll}
A \mid\equiv \overset{K_a}{\mapsto} A & B \mid\equiv \overset{K_b}{\mapsto} B \\
A \mid\equiv \overset{K_b}{\mapsto} B & B \mid\equiv \overset{K_a}{\mapsto} A \\
A \mid\equiv \sharp(N_a) & B \mid\equiv \sharp(N_b) \\
A \mid\equiv A \overset{R_a}{\rightleftharpoons} B & B \mid\equiv A \overset{R_b}{\rightleftharpoons} B \\
A \mid\equiv B \mid\Rightarrow A \overset{R_b}{\rightleftharpoons} B & B \mid\equiv A \mid\Rightarrow A \overset{R_a}{\rightleftharpoons} B
\end{array}$$

Both principal A and B know their own private and public keys. In addition A and B know the public keys of each other. A believes that the nonce N_a is fresh. B believes that the nonce N_b is fresh. A and B believe that the key components R_a R_b are secrets they shared with each other. A believes that R_a is a good session key component for communication with B, and B believes that $R_b a$ is a good session

[5]The delegation certificate is omitted here. It is only used to remove one level of indirection in the authentication path.

key component for communication with A. Finally, both parties trust the other party to generate proper session key components.

The protocol analysed. Message 1 (`HELLO`) is sent from A to B to initiate the authentication and key negotiation protocol:

$$B \triangleleft \{N_a\}_{K_a^{-1}} \tag{7.3}$$

Using its knowledge about A's public key and the *message-meaning rule*, B believes that N_a was once sent by A:

$$B \mid\equiv A \mid\sim N_a \tag{7.4}$$

Since Message 1 does not contain anything B knows is fresh, it cannot determine whether the message is a replay. Thus, there is no way to make the **said** ($\mid\sim$) statement into a **believes** ($\mid\equiv$) statement. This is also why the SCIP protocol can not be classified as a fail-stop, but rather as fail-safe (see Section 7.5.1). B has no other choice but to respond to the message as if it is fresh, or terminate the protocol to satisfy the fail-stop property by not sending a respond message.

The less restrictive fail-safe protocol condition will allow B to respond to Message 1. As it was shown in Section 7.5.2, this does not violate the secrecy assumption. Message 2 (`RESPOND`) is sent from B to A to transfer B's session key component R_b to A:

$$A \triangleleft \{\{N_a, N_b,\ A \overset{R_b}{\rightleftharpoons} B\}_{K_b^{-1}}\}_{K_a} \tag{7.5}$$

Using its own secret key K_a^{-1}, A can decrypt the message and obtain:

$$A \triangleleft \{N_a, N_b,\ A \overset{R_b}{\rightleftharpoons} B\}_{K_b^{-1}} \tag{7.6}$$

From the *message-meaning rule* for public keys A can deduce:

$$A \mid\equiv B \mid\sim (N_a, N_b, A \overset{R_b}{\rightleftharpoons} B) \tag{7.7}$$

Since A believes that N_a is fresh, A can also deduce:

$$A \mid\equiv \sharp(N_a, N_b,\ A \overset{R_b}{\rightleftharpoons} B) \tag{7.8}$$

From the *nonce verification rule* A can derive:

$$A \mid\equiv B \mid\equiv (N_a, N_b,\ A \overset{R_b}{\rightleftharpoons} B) \tag{7.9}$$

and

$$A \mid\equiv B \mid\equiv A \overset{R_b}{\rightleftharpoons} B \tag{7.10}$$

Since A believes that B controls $A \overset{R_b}{\rightleftharpoons} B$, it can use the the *jurisdiction rule* to deduce:

$$A \mid\equiv A \overset{R_b}{\rightleftharpoons} B \tag{7.11}$$

Thus, A is now in a position to believe that it shares the secret R_b with B.

Message 3 (`FINISH`) is sent from A to B to transfer A's session key component R_a to B:

$$B \triangleleft \{\{A \overset{R_a}{\rightleftharpoons} B,\ N_b, \{D\}_{K_{ab}}\}_{K_a^{-1}}\}_{K_b} \tag{7.12}$$

Using the same reasoning as for the `RESPOND` message, and using N_b as freshness identifier instead of N_a, the following is derived:

$$B \mid\equiv A \overset{R_a}{\rightleftharpoons} B \tag{7.13}$$

Thus, at the end of the protocol run, both A and B are entitled to believe:

$$A \overset{R_b}{\rightleftharpoons} B \textit{ and } A \overset{R_a}{\rightleftharpoons} B \tag{7.14}$$

Reasoning about the new session key. Further reasoning about the session key components cannot be carried out using the original BAN logics. BAN has no rules that can be used to reason about the the two session key components R_a and R_b and how they eventually will become the new session key.

To reason about the new session key, the following BAN-like postulate for splicing secrets is introduced:

$$\frac{P \mid\equiv P \overset{x}{\rightleftharpoons} Q,\ P \mid\equiv P \overset{y}{\rightleftharpoons} Q}{P \mid\equiv P \overset{(x \oplus y)}{\rightleftharpoons} Q}$$

where $\oplus$ denotes a function of two arguments known to P and Q. Applying this postulate the following is obtained:

$$A \mid\equiv A \overset{R_a \oplus R_b}{\rightleftharpoons} B$$

$$B \mid\equiv A \overset{R_a \oplus R_b}{\rightleftharpoons} B$$

If K_{ab} (session key) is defined as: $K_{ab} = R_a \oplus R_b$ and $K_{ab}s$ is subsequently used as a shared key, and thus:

$$A \mid\equiv A \overset{K_{ab}}{\leftrightarrow} B$$

$$B \mid\equiv A \overset{K_{ab}}{\leftrightarrow} B$$

Thus, both A and B are entitled to believe that they possess the new session key, and can use it to encrypt further communication between the two of them.

The result of a successful execution of the SCIP protocol is that both hosts believe they have a new session key that can be used to communicate with the remote host. A property that the current protocol achieves partially, is that each host believes that the other host also believes in the new key. The following:

$$A \mid\equiv B \mid\equiv A \overset{K_{ab}}{\leftrightarrow} B$$

$$B \mid\equiv A \mid\equiv A \overset{K_{ab}}{\leftrightarrow} B$$

is not established immediately. The last two statements are established as soon as an IP datagram, encrypted with the new session key has been decrypted successfully by both parties. By decrypting D with the session key, B can establish A's belief in the new session key, but A is not in the same position. In the current design, this is considered acceptable—simply because, in the implementation, no datagrams will be handed over to the higher level protocols before they have been decrypted successfully.

7.6. Discussion

This section discusses some of the design choices made during the development of the SCIP system. The discussion focuses on the architecture, the use of a fail-stop methodology to develop the SCIP protocol, and compares SCIP with swIPe.

The use of a user-space process to implement the policy engine reduced the complexity of debugging the system significantly. It is also relatively easy to experiment with different algorithms and protocols. The initial prototype is still inefficient—simply because all communication using the SCIP protocol passes through *scipd*. Consequently, all IP datagrams are copied across the kernel/user space boundary twice both upon reception and transmission. However, a kernel implementation of the encryption engine eliminates most of the overhead. More tasks are then be carried out in the kernel, and the need for data copying is reduced.

As shown Chapter 3, security functions can be added to a networked system in two different ways. The first approach is application-aware and provides end-to-end security at the application level. The other approach is not application aware and provides security functions at the network-level.

The advantage of network-level security is that it can be added to a system and be used without any modifications to the applications at all. The disadvantage is that security arguments appear at the host level and not on a per application basis. This means that the granularity is significantly reduced. The Taos operating system [Wobber et al., 1994] contains a novel way to multiplex a link-level encryption channel between several applications. However, the granularity is still at the host level, and the operating system must, of course, always be trusted to authenticate local inter-process channels properly.

The single most difficult problem of applying a fail-stop approach to secure protocol design is to reason about the freshness of messages. Indeed, this was what fail-stop protocol design should solve in the first place by always require that messages contain sufficient information to be bound together in a protocol execution.

In pure nonce based protocols there is nothing that the recipient of the first message can determine as fresh unless it has provided the sender with something in advance. When time-stamps are used, a recipient will not be able to distinguish between an authentic message and a replay that occurs within the time interval in which the message, according to a protocol specification, should be considered fresh.

Although the fail-stop approach to secure protocol design seems good in theory, most protocols developed using this approach will be fail-safe. It can be argued that the delegation certificate embedded in the first message of the SCIP protocol should be considered a time-stamp freshness identifier, and that due to this, the protocol should also satisfy the fail-stop condition. However, it is obvious that the recipient of this message needs to apply knowledge about the sender in order to do any reasoning about the freshness of the message. The delegation only says that the

sender was booted correctly recently. Another solution is, as it was noted on in Section 3.7.3, to rely on a trusted third party providing a beacon.

In comparison to swIPe, both performance and functionality of SCIP differ from that of swIPe. SwIPe mainly implements a mechanism for tunnelling of IP traffic and a minimal implementation of policies. The main goal of SCIP is on implementation of policies, in particular on authentication and key negotiation and access control. These issues are not solved in the original swIPe implementation, but solutions have been incorporated into other systems, such as, SKIP and the IP Next Generation framework.

Since swIPe implements encryption functions in the kernel, its design is inherently more efficient than that of SCIP. However, since SCIP facilitates the use of a faster encryption algorithm, the actual performance difference is minimal. A speed-up similar to that of swIPE would also be achieved with SCIP when the encryption functions are moved into the kernel.

7.7. Summary

In this chapter, the design and implementation of SCIP, a secure network layer under Unix is described and some of the performance penalties of using the system are discussed. SCIP is a network level authentication and key distribution system that authenticates two communicating parties mutually, and exchange session key components that both parties use to build a new session key.

SCIP is still a prototype implementation and performance is expected to increase significantly when the encryption engine is migrated into the Unix kernel. The authentication and key negotiation protocol protocol satisfies the fail-safe property and is acceptable for its intended usage. The use of a fail-stop design for secure protocol simplified the protocol development and its verification.

—8—

Summary and Concluding Remarks

This chapter summarises the results and findings of this dissertation and presents its conclusions. Directions of future research are also outlined.

8.1. Summary

In this dissertation, the security of end systems hosting electronic-commerce applications has been investigated and a system architecture for secure user-controlled electronic transactions has been proposed. The primary goal of the research has been to determine the factors affecting security in such systems, and to derive a system architecture for new generations of personal computers.

Chapter 2 surveyed the field of cryptography and described the most commonly used cryptographic techniques. The legal aspects of applying cryptography in civilian appliances were also discussed.

Chapter 3 surveyed the fundamentals of computer and network security and focused on security models and design methodologies underlying secure computer systems. The chapter introduced the basics of computer security and defined the terms that were used in the remainder of this dissertation.

Chapter 4 presented the Trust architecture for end systems targeted at hosting electronic-commerce applications. The Trust architecture provides a trusted path between user and their signing keys, and incorporates security functions into the human-visible computer interface. The architecture supports interaction between the user and a device that enables a transaction to be conducted securely between that device and another system. The transaction is not conducted by the device unless it is explicitly authorized by the user to do so. The operating system and untrusted helper applications that communicate with remote computer systems are not part of the system's TCB.

Chapter 5 discussed the use of transaction protocols and logging facilities in electronic-commerce applications with the main focus on protocol requirements and specification. An example protocol for contract signing was also presented.

Chapter 6 described how authorisation and access control is achieved in μ-CAP. The μ-CAP architecture has been designed to enable users to define their own access control policies and have them enforced by remote servers. In μ-CAP, access rights can be propagated between users and exercised by the server without any communication with the object server prior to access requests.

Chapter 7 described the design and implementation of a network-level security system, and how this system protects communicating end systems against active and passive attacks from a third party.

8.2. Concluding Remarks

In other disciplines of computer science, terms such as *optimal* and *lower bound*, have been used to express the efficiency of a solution when being compared to known theoretical limits (see, e.g., [Knuth, 1973]). The level of security in a system, however, is related more to robustness than to performance.

In secure system design, there are many tradeoffs to take into account [Meadows, 1994]: different requirements must be traded off against each other. In this respect, secure-systems engineering is no different from other engineering processes.

The system architecture described in this dissertation can be used to increase the level of security in a personal computing environment. In this respect, the results and findings of this dissertation represent information that can be used to increase the security of electronic commerce systems.

A personal computer is more than just a small computer used by one person at a time. Small, portable machines are like traditional personal organizers and they contain a wealth of private information. If users are to incorporate them into accessing distributed services in the Internet, they must be confident that their computers can be trusted to act on their behalf and not compromise any private information.

Small, personal machines will contain more and more personal information, and they will be seen as the manifestation of the user in security-demanding applications such as electronic commerce. Privacy and security of communication and storage are therefore paramount. In addition, the confidentiality of signing keys and the integrity of the information presented to the user are equally important factors.

A user's security depends on the proper functioning of a personal computer to protect critical information (for example, signing keys used for digital signatures). The user is the principal figure, and access to information in the personal computer must not be possible without its owner's permission. The personal computer is the tool used to engage in electronic transactions securely.

Any tool can be difficult to use, computers especially so, and cryptography has been very badly used even by experienced computer scientists. Roger Needham remarked that "Whenever anyone says that a problem is easily solved by cryptography, it shows that he doesn't understand it" [Needham, 1993]. However, secure communication and transaction processing depend heavily on both trusted systems and users who make "the right" decisions. Unless a computer system is trustworthy it is questionable whether it can mediate information precisely to the user. If information is not presented correctly to the human user, the user makes wrong decisions.

[Brooks, 1996] argued that getting information from a machine into a human's head is the central task of computer graphics. A personal computer conducting transactions with remote services on behalf of the user can be considered a joint

human-computer controlled system. Therefore, the interface between the user and the computer plays a vital rôle. After all, it is the human user who must make the final decision of whether to authorise a transaction, and it is important that valid information is presented to the user.

One of the most problematic threats is nonrepudiation. In communication with a remote service, a user must be able to trust that the owner/organisation of the remote machine cannot successfully claim that his system was compromised. For this reason, it is not only important that the user's own key management is secure, but also that that the other principal must have key management that is trustworthy enough that, if his keys would have been compromised, it could only have happened through gross negligence. In non-electronic commerce, negligence is something that a party can be held accountable for. If a bank makes a mistake, it should be held liable, and the same should hold for an opponent in a disputed transaction.

[Thompson, 1984] points out that code written by others cannot be trusted. A well-installed Trojan horse will be almost impossible to detect even with source-level verification and scrutiny. This is bad news for security and for users who want to establish trust in their personal computers. However, in the presented design, this does not matter fundamentally. The main issue is the *integrity* of information presented to the user and a relatively simple hardware mechanism to enforce this functionality. No assumptions are made — or need to be made — about the security of other helper applications that the user might use (including word processors, file systems, compilers, WWW browser, etc.).

Untrusted helper applications that either implement some security functions on their own (for example, WWW browsers) or rely on the system to provide such services will be commonplace. In this dissertation, two such helper applications have been designed: the μ-CAP authorisation and access control architecture, and the SCIP architecture for secure network-level communication. Both these applications implement security functions that in other systems are typically located inside the TCB of the system in question.

In the days when centralized non-networked mainframe computers were commonly in use, an approach to security was to lock up the mainframe computer and its terminals inside a physically secure area that could only be accessed by authorised personnel. By using the system architecture described in this dissertation, a similar advantage is achieved for personal computers in networked environments. Vital information is protected against unauthorised access from networked principals.

In this dissertation, a computer system that provides a trusted path between a user and his signing keys, and incorporates security functions into the human-computer interface has been presented. The architecture supports interaction between the user and a device that enables a transaction to be conducted securely between that device and another system. The transaction is not conducted by the device unless it is explicitly authorized by the user to do so.

The fundamental property of the Trust architecture is the division between a secure subsystem consisting of a smart card and terminal I/O system, and an untrusted component consisting of the main CPU, memory and other devices.

For the correct operation of the security module, it is essential that the security module can gain exclusive control over the communication of the input/output device, the interaction with the (optional) smart card, and the communication with the local operating system. Controlling the transition between secure and insecure mode of operation is also essential to the operational behaviour of the system.

The main conclusion of this dissertation is that the security of end systems hosting electronic-commerce applications, crucially depends on the presence of a trusted path between the user of the system and the signing keys used to authorise statements on behalf of that user. One way to achieve this, is by incorporating the input and output devices of the human-visible computer interface into the trusted computing base of the system.

However, a trusted path between the users and their signing keys is not sufficient to safeguard against all kinds of attacks on the integrity of information presented to them. The human-computer interfaces would also have to be resistant against integrity threats, and safeguard the procedure under which the human user can issue authorisation statements. Both these properties are essential in order to prevent nonrepudiation.

Authorisation and access control decision should eventually be taken by a human user. The user constitutes the true end point of communication. It is fundamentally important for electronic-commerce systems to keep users in the control loop. Application programs and computer systems are merely proxies acting on behalf of users. The fundamental issues are to bind humans to signing keys, and provide adequate protection for these keys.

A secure end system enables a human user to respond more securely to information presented to him via the human-computer interface. By requiring that authorisations should be explicit by prompting for them, and by ensuring that the implementation of the underlying authorisation mechanisms cannot be subverted, increased confidence in the operational functionality of the system is obtained.

The system architecture described in Chapter 4 is designed to provide a trusted path between a user and his signing keys, and to preserve the integrity of the human-visible user interface.

Untrusted "helper applications" with security requirements executing outside the TCB (such as those mentioned in Chapter 6 and Chapter 7) can be used in conjunction with the proposed system design. In this dissertation, two such applications have been designed and analysed: the μ-CAP authorisation and access control system, and the SCIP architecture and network-level authentication and key distribution protocol. μ-CAP's functionality can be used to transfer trust between principals in the system. SCIP can be used to establish secure communication channels between pairs of end systems at the network level.

During the initial design phase of the Trust architecture, it was suspected that secure hardware would have to play an important rôle in the final system

architecture, but it was not clear what system functionality would require such hardware support. Studies in the area of system safety and computers indicated that the use of hardware mechanism that can not be overridden by software have been used with much success in that area to provide safer user interfaces. In a wider context, the conclusions of this dissertation indicate that it is doubtful whether a secure personal computer can be built without using such functionality. In addition, the use of smart-card related technologies without other mechanisms to provide a trusted path between human users and their signing keys, will at best only be able to protect the signing keys from being disclosed, but will not be able to protect against malicious software attacking other vital functions of a transaction system.

8.3. Future research

In this dissertation, several simplifications have been made in modelling human users. In particular, not much attention has been paid to cognitive-ergonomic design issues. An interesting issue is to investigate what characteristics a good human-computer interface for electronic commerce has. Clearly, and as it has been constantly pointed out in this dissertation, it is essential to keep the human user in the control loop of authorisation. Further research would benefit from studies of tasks/scenarios with human users participating in electronic transactions.

Research and development of transaction protocols, arbitration, and secure user interfaces will be continued in the Moby Dick project, and in particular, in the design and implementation of a portable handheld computer with wireless communication facilities. Issues involved in the integration of the proposed design with existing payment protocols is also a subject for further investigation.

In a joint project with the University of Tromsø, Norway, the design of the μ-CAP authorisation and access control architecture for personal computing environments is continued. Although there are other issues of concern for that research, it is expected that the μ-CAP protocols will be used in the project.

Bibliography

[Abadi et al., 1990] Abadi, M., Burrows, M., Kaufman, C., and Lampson, B. (1990). Authentication and Delegation with Smart-cards. Technical Report 67, Digital Systems Research Center, 130 Lytton Avenue, Palo Alto, California 94301.

[Abadi et al., 1992] Abadi, M., Burrows, M., Lampson, B., and Plotkin, G. (1992). A Calculus for Access Control in Distributed Systems. In *Advances in Cryptology—Crypto'91*, pages 1–23. Springer-Verlag.

[Abadi and Needham, 1994] Abadi, M. and Needham, R. M. (1994). Prudent Engineering Practice for Cryptographic Protocols. Technical Report 125, Digital Systems Research Center, 130 Lytton Avenue, Palo Alto, California 94301.

[Abadi and Tuttle, 1991] Abadi, M. and Tuttle, M. (1991). A Semantics for a Logic of Authentication. In *Proceedings of the 10th Annual ACM Symposium on Principles of Distributed Computing*, pages 201–216.

[Abelson et al., 1997] Abelson, H., Anderson, R. J., Bellovin, S. M., Benaloh, J., Blaze, M., Diffie, W., Gilmore, J., Neumann, P. G., Rivest, R. L., Schiller, J. I., and Schneier, B. (1997). The Risks of Key Recovery, Key Escrow, and Trusted Third-Party Encryption. *The World Wide Web Journal*, 2(3):241–257.

[Accetta et al., 1986] Accetta, M., Baron, R., Bolosky, W., Golub, D., Rashid, R., Tevanian, A., and Young, M. (1986). Mach: A New Kernel Foundation for UNIX Development. In *USENIX Summer'86 Conf. Proc.*, pages 93–112.

[Amoroso, 1994] Amoroso, E. (1994). *Fundamentals of Computer Security Technology*. Prentice Hall.

[Anderson and Needham, 1995a] Anderson, R. and Needham, R. M. (1995a). Programming Satan's Computer. In van Leeuwen, J., editor, *Computer Science Today — Recent Trends and Developments*, volume 1000 of *Lecture Notes in Computer Science*, pages 426–440. Springer-Verlag.

[Anderson and Needham, 1995b] Anderson, R. and Needham, R. M. (1995b). Robustness Principles for Public Key Protocols. In *Advances in Cryptology — Proceedings of Crypto '95*, volume 963 of *Lecture Notes in Computer Science*, pages 236–247. Springer-Verlag.

[Anderson, 1993] Anderson, R. J. (1993). The Classification of Hash Functions. In *'Codes and Ciphers', proceedings of Fourth IMA Conference on Cryptography and Coding*, pages 83–93.

[Anderson, 1994a] Anderson, R. J. (1994a). Liability and Computer security: Nine principles. *Computer Security — ESORICS 94, Springer*, 875:231–245.

[Anderson, 1994b] Anderson, R. J. (1994b). Why Cryptosystems fail. *Communications of the ACM*, 37(11):32–40.

[Anderson, 1995] Anderson, R. J. (1995). Crypto in Europe — Markets, Law and Policy. In *Cryptographic Policy and Algorithms*, Queensland.

[Anderson and Biham, 1995] Anderson, R. J. and Biham, E. (1995). Tiger: A Fast New Hash Function. In *Fast Software Encryption*, volume 1039 of *Lecture Notes in Computer Science*, pages 89–97. Springer-Verlag.

[Anderson and Kuhn, 1996] Anderson, R. J. and Kuhn (1996). Tamper Resistance — a Cautionary Note. In *Proceedings of the 2nd Workshop On Electronic Commerce*, Oakland, California. USENIX Association.

[ANSI X3.106, 1983] ANSI X3.106 (1983). American National Standard for Information Systems—Data Encryption Algorithm—Modes of Operation. American National Standards Institute.

[Atkins et al., 1995] Atkins, D., Graff, M., Lenstra, A. K., and Leyland, P. C. (1995). THE MAGIC WORDS ARE SQUEAMISH OSSIFRAGE. In *Proceedings Asiacrypt'94*, pages 263–277. Lecture Notes in Computer Science 917.

[Atkinson, 1995a] Atkinson, R. (1995a). IP Authentication header. RFC 1826.

[Atkinson, 1995b] Atkinson, R. (1995b). IP Encapsulating Security Payload. RFC 1827.

[Atkinson, 1995c] Atkinson, R. (1995c). IP Security Architecture. RFC 1825.

[Badrinath et al., 1993] Badrinath, B. R., Acharya, A., and Imielinski, T. (1993). Impact of Mobility on Distributed Computations. *ACM Operating Systems Review*, 27(2):15–20.

[Bauer et al., 1983] Bauer, R. K., Berson, T. A., and Feiertag, R. J. (1983). A Key Distribution Protocol using Event Markers. *ACM Transactions on Computer Systems*, 1(3):249–255.

[Bell and LaPadula, 1973] Bell, D. and LaPadula, L. (1973). Secure Computer Systems: Mathematical Foundations. Technical Report ESD-TR-73-278, Vol. I, Mitre Corporation.

[Bellovin, 1990] Bellovin, S. M. (1990). Pseudo-Network Drivers and Virtual Networks - Extended Abstract. In *USENIX Conference Proceedings*, pages 229–244.

[Bellovin and Merrit, 1991] Bellovin, S. M. and Merrit, M. (1991). Limitations of the Kerberos authentication system. In *USENIX Conference Proceedings*, pages 253–267, Dallas.

[Biham and Shamir, 1991] Biham, E. and Shamir, A. (1991). Cryptanalysis of DES-like Cryptosystems. In *Advances in Cryptology—Proceedings of Crypto'90*, pages 2–21. Springer-Verlag.

[Birrel and Nelson, 1984] Birrel, A. and Nelson, B. (1984). Implementing Remote Procedure Calls. *ACM Transactions on Computer Systems*, 2:39–59.

[Black, 1991] Black, A. P. (1991). Understanding Transactions in the Operating System Context. *ACM Operating Systems Review*, 25(1):73–76.

[Blaze, 1994a] Blaze, M. (1994a). Key Management in an Encrypting File System. In *USENIX Summer 1994 Technical Conference*, Boston, MA.

[Blaze, 1994b] Blaze, M. (1994b). Protocol Failure in the Escrowed Encryption Standard. In *Proceedings of Second ACM Conference on Computer and Communications Security*, Fairfax, VA.

[Blaze et al., 1996a] Blaze, M., Diffie, W., Rivest, R. L., Schneier, B., Shimomura, T., Thompson, E., and Wiener, M. (1996a). Minimal Key Length for Symmetric Ciphers to Provide Adequate Commercial Security. A Report by an Ad Hoc Group of Cryptographers and Computer Scientis.

[Blaze et al., 1996b] Blaze, M., Feigenbaum, J., and Lacy, J. (1996b). Decentralized Trust Management. In *IEEE Conference on Security and Privacy*, Oakland, CA.

[Bradner and Mankin, 1995] Bradner, B. and Mankin, A. (1995). The Recommendation for the IP Next Generation Protocol. RFC 1752.

[Brewer et al., 1995] Brewer, E., Gauthier, P., Goldberg, I., and Wagner, D. (1995). Basic Flaws in Internet Security and Commerce. `http://http.cs.berkeley.edu/~gauthier/-endpoint-security.html`.

[Brooks, 1996] Brooks, F. P. (1996). The Computer Scientist as Toolsmith II. *Communications of the ACM*, 39(3):61–68.

[Burrows et al., 1991] Burrows, M., Abadi, M., and Needham, R. (1991). A Logic of Authentication. Technical Report 39, Digital Systems Research Center, 130 Lytton Avenue, Palo Alto, California 94301.

[Canadian System Security Centre, 1993] Canadian System Security Centre (1993). The Canadian Trusted Computer Product Evaluation Criteria (CTCPEC).

[Caronni et al., 1996] Caronni, G., Lubich, H., Aziz, A., Markson, T., and Skrenta, R. (1996). SKIP — Securing the Internet. In *Proceedings of the Fifth Workshop on Enabling Technologies, (WET ICE '96)*. IEEE Computer Society Press.

[CCITT, 1991] CCITT (1991). Information Technology — Open Systems Interconnection — The Directory: Authentication Framework. CCITT Recommodation X.509, ISO/IEC 9594-8.

[Cheswick, 1992] Cheswick, W. R. (1992). An Evening With Berferd, in Which a Cracker is Lured, Endured, and Studied. In *Proceedings Winter USENIX Conference*. San Francisco, CA.

[Cheswick and Bellovin, 1994] Cheswick, W. R. and Bellovin, S. M. (1994). *Firewalls and Internet Security, Repelling the Wily Hacker*. Addison Wesley.

[Clark and Wilson, 1987] Clark, D. and Wilson, D. (1987). A Comparison of Commercial and Military Computer Security Policies. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 184–194, Oakland, CA, USA. IEEE Computer Society Press.

[Coulouris et al., 1994] Coulouris, G., Dollimore, J., and Kindberg, T. (1994). *Distributed Systems, Concepts and Design*. Addison-Wesley.

[de Viva et al., 1995] de Viva, M., de Viva, G. O., and Gonzales, L. (1995). A Brief Essay on Capabilities. *ACM Sigplan Notices*, 30(7):29–36.

[Denning, 1982] Denning, D. E. (1982). *Cryptography and Data Security*. Addison-Wesley, Reading, Mass.

[Denning, 1993] Denning, D. E. (1993). To Tap Or Not To Tap. *Communications of the ACM*, 36(3):24–33.

[Denning and Branstad, 1996] Denning, D. E. and Branstad, D. K. (1996). A Taxonomy for Key Escrow Encryption Systems. *Communications of the ACM*, 39(3):34–40.

[Denning and Sacco, 1981] Denning, D. E. and Sacco, G. M. (1981). Timestamps in Key Distribution Protocols. *Communications of the ACM*, 24(8):533–536.

[Dennis and van Horn, 1966] Dennis, J. B. and van Horn, E. C. (1966). Programming Semantics for Multiprogrammed Computations. *Communications of the ACM*, 9(3):143–155.

[Department of Defense, 1985] Department of Defense (1985). DoD 5200.28-STD: Department of defense (DoD) Trusted Computer System Evaluation Criteria (TCSEC).

[Diffie and Hellmann, 1976] Diffie, W. and Hellmann, M. E. (1976). New Directions in Cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654.

[Diffie and Hellmann, 1977] Diffie, W. and Hellmann, M. E. (1977). Exhaustive Cryptanalysis of the NBS Data Encryption Standard. *Computer*, 10:74–84.

[Eberle, 1992] Eberle, H. (1992). A High-speed DES Implementation for Network Applications. Technical Report 90, Digital Systems Research Center, 130 Lytton Avenue, Palo Alto, California 94301.

[Federal Bureau of Investigation, 1992] Federal Bureau of Investigation (1992). Digital Telephony. Technical report, United States Department of Justice.

[Felten et al., 1996] Felten, E. W., Balfanz, D., Dean, D., and Wallach, D. S. (1996). Web Spoofing: An Internet Con Game. Technical Report 540-96, Department of Computer Science, Princeton University.

[Finin, 1989] Finin, T. (1989). GUMS – a general user modelling shell. In Wahlster, W. and Kobsa, A., editors, *User Models in Dialog Systems*. Springer-Verlag, Berlin.

[Freier et al., 1996] Freier, A. O., Karlton, P., and Kocher, P. C. (1996). The SSL Protocol, Version 3.0. Internet draft (`http://home.netscape.com/eng/ssl3/ssl-toc.html`). Expires 9/96.

[Gong, 1989a] Gong, L. (1989a). A Secure Identity-Based Capability System. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 56–63, Oakland, California.

[Gong, 1989b] Gong, L. (1989b). On Security in Capability-Based Systems. *ACM Operating Systems Review*, 23(2):56–60.

[Gong, 1990] Gong, L. (1990). *Cryptographic Protocols for Distributed Systems*. PhD thesis, Cambridge University.

[Gong, 1992] Gong, L. (1992). A Security Risk of Depending on Synchronized Clocks. *ACM Operating Systems Review*, 26(1).

[Gong, 1995] Gong, L. (1995). Efficient Network Authentication Protocols: Lower Bounds and Optimal Implementations. *Distributed Computing*, 9(3).

[Gong et al., 1993] Gong, L., Lomas, M. A., Needham, R. M., and Saltzer, J. H. (1993). Protecting Poorly Chosen Secrets from Guessing Attacks. *IEEE Journal on Selected Areas in Communications*, 11(5):648–656.

[Gong et al., 1990] Gong, L., Needham, R., and Yahalom, R. (1990). Reasoning about Belief in Cryptographic Protocols. In *Proceedings of the IEEE 1990 Symposium on Security and Privacy*, pages 234–248, Oakland, California.

[Gong and Syverson, 1995] Gong, L. and Syverson, P. (1995). Fail-Stop Protocols: An Approach to Designing Secure Protocols. In *Proceedings of the 5th IFIP Working Conference on Dependable Computing for Critical Applications*, Urbana-Champaign, Illinois.

[Haber and Stornetta, 1991] Haber, S. and Stornetta, W. S. (1991). How to Time-stamp a Digital Document. *Journal of Cryptology*, 3(2):99–111.

[Haller, 1994] Haller, N. M. (1994). The S/KEY One-time Password System. In *Proceedings of the Internet Society Symposium on Network and Distributed System Security*, San Diego, CA.

[Härder and Reuter, 1983] Härder, T. and Reuter, A. (1983). Principles of Transaction-Oriented Database Recovery. *Computing Surveys*, 15(4).

[Hellman et al., 1976] Hellman, M. E., Merkle, R., Schroeppel, R., Washington, L., Diffie, W., Pohlig, S., and Scheweitzer, P. P. (1976). Results of an Initial Attempt to Cryptanalyze the NBS Data Encryption Standard. Technical Report SEL 76-042, Information Systems Lab, Department of Electrical Engineering, Stanford University.

[Helme and Plaggenmarsch, 1995] Helme, A. and Plaggenmarsch, E. (1995). SCIP: A Secure Network Layer under Unix. Technical Report 95–06, University of Twente, The Pegasus group.

[Helme and Stabell-Kulø, 1996] Helme, A. and Stabell-Kulø, T. (1996). Off-Line delegation in a File Repository. In *1996 DIMACS WorkShop on Trust Management in Networks*, Rutgers University. Work presented at workshop, but no proceedings.

[Helme and Stabell-Kulø, 1997] Helme, A. and Stabell-Kulø, T. (1997). Security Functions for a File Repository. *ACM Operating Systems Review*, 31(2):3–8.

[Hinden, 1996] Hinden, R. M. (1996). IP Next Generation Overview. *Communications of the ACM*, 39(6):61–71.

[Ioannidis and Blaze, 1993] Ioannidis, J. and Blaze, M. (1993). The Architecture and Implementation of Network-layer security under Unix. In *Proceedings of the Fourth Usenix Unix Security Symposium*, pages 29–39.

[ITSEC, 1991] ITSEC (1991). Information Technology Security Evaluation Criteria (ITSEC). Printed and published by the Department of Trade and Industry, London. Harmonised Criteria of France, the Netherlands, and the United Kingdom.

[Jacobsen, 1991] Jacobsen, V. (1991). Compressing TCP/IP Headers for Low-Speed Links. RFC 1144.

[Jaffe, 1988] Jaffe, M. S. (1988). *Completeness, Robustness, and Safety of Real-Time Requirements Specification*. PhD thesis, University of California, Irvine, California.

[Jones, 1978] Jones, A. K. (1978). Protection Mechanisms and the Enforcement of Security Policies. In Bayer, R., Graham, R. M., and Seegmüller, G., editors, *Operating Systems: an Advanced Course*, volume 60 of *Lecture Notes in Computer Science*, pages 229–251. Springer-Verlag, Berlin.

[Kahn, 1967] Kahn, D. (1967). *The Code-Breakers*. Macmillan, New York.

[Kent, 1996] Kent, S. (1996). Let A Thousand (Ten Thousand?) CAs Reign. 1996 DIMACS Workshop on Trust Management in Networks.

[Knuth, 1973] Knuth, D. E. (1973). *The Art of Computer Programming. Vol. 3: Sorting and Searching*. Addison-Wesley.

[Koops, 1996] Koops, B.-J. (1996). A Survey of Cryptography Laws and Regulations. *The Computer Law & Security Report*, pages 349–355.

[Lai, 1991] Lai, X. (1991). Detailed Description and a Software Implementation of the IPES Cipher. Institute for Signal and Information Processing, ETH-Zentrum, Zurich, Switzerland.

[Lamport, 1981] Lamport, L. (1981). Password Authentication with Insecure Communication. *Communications of the ACM*, 24(11):770–772.

[Lampson, 1971] Lampson, B. (1971). Protection. In *Proceedings Fifth Princeton Symposium on Information Sciences and Systems*, pages 437–443, Princeton University. Reprinted in Operating Systems Review, 8, 1, January 1974, pp. 18–24.

[Lampson, 1984] Lampson, B. (1984). Hints for Computer System Design. *IEEE Software*, 1(1):11–31.

[Lampson et al., 1992a] Lampson, B., Abadi, M., Burrows, M., and Wobber, E. (1992a). Authentication in Distributed Systems: Theory and Practice. Technical Report 83, Digital Systems Research Center, 130 Lytton Avenue, Palo Alto, California 94301.

[Lampson et al., 1992b] Lampson, B., Abadi, M., Burrows, M., and Wobber, E. (1992b). Authentication in Distributed Systems: Theory and Practice. *ACM Transactions on Computer Systems*, 10(4):265–310.

[Landau et al., 1994] Landau, S., Kent, S., Brooks, C., Charney, S., Denning, D., Diffie, W., Lauck, A., Miller, D., Neumann, P., and Sobel, D. (1994). Codes, Keys and Conflicts: Issues in U.S. Crypto Policy. Technical Report ACM ISBN: 0-89791-677-8, Association for Computing Machinery, Inc., 1515 Broadway, New York, NY 10036. Report of a Special Panel of the ACM U.S. Public Policy Committee (USACM).

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

[Lenstra, 1996] Lenstra, A. K. (1996). New Factorization Record. Usenet posting to `sci.crypt.research`.

[Lenstra and Lenstra, 1993] Lenstra, A. K. and Lenstra, H. W. J. (1993). *The Development of the Number Field Sieve*. Springer-Verlag.

[Leveson, 1995] Leveson, N. G. (1995). *SAFEWARE: System Safety and Computers*. Addison-Wesley.

[Liebl, 1993] Liebl, A. (1993). Authentication in Distributed Systems: A Bibliography. *ACM Operating Systems Review*, 27(4):31–41.

[Linn, 1993a] Linn, J. (1993a). Generic Security Service Application Program Interface. RFC 1508.

[Linn, 1993b] Linn, J. (1993b). Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures. RFC1421, Internet Network Working Group.

[Lomas et al., 1989] Lomas, M. A., Gong, L., Saltzer, J. H., and Needham, R. M. (1989). Reducing Risks from Poorly Chosen Keys. In *Proceedings 12th ACM Symposium on Operating Systems Principles*, pages 14–18.

[Lund, 1996] Lund (1996). Lund-rapporten (Dokument nr. 15). Report to Stortinget from the commission set down by Stortinget to investigate claims about illegal surveillance of Norwegian citizens.

[Matsui, 1994] Matsui, M. (1994). Linear Cryptanalysis Method for DES Cipher. In *Advances in Cryptology—Proceedings EUROCRYPT'93*, pages 386–397. Springer-Verlag.

[McKusick et al., 1996] McKusick, M. K., Bostic, K., Karels, M. J., and Quarterman, J. S. (1996). *The Design and Implementation of the 4.4BSD Operating System*. Addison-Wesley.

[Meadows, 1994] Meadows, C. (1994). Tradeoffs in Secure System Development: An Outline. In *Proceedings of CSESAW '94*. Naval Surface Warfare Center.

[Meadows, 1995] Meadows, C. (1995). Formal Verification of Cryptographic Protocols: A Survey. In *Advances in Cryptology – Asiacrypt '9*, number 917 in Lecture Notes in Computer Science, pages 133–150. Springer-Verlag.

[Modugno et al., 1996] Modugno, F., Leveson, N. G., Reese, J. D., Partridge, K., and Sandys, S. (1996). Creating and Analyzing Requirement Specifications of Joint Human-Computer Controllers for Safety-Critical Systems. In *1996 Symposium on Human Interaction with Complex Systems (HICS'96)*, Dayton, OH.

[Morris et al., 1986] Morris, J., Satyanarayanan, M., Conner, M. H., Howard, J. H., Rosenthal, D. S., and Smith, F. D. (1986). Andrew: a Distributed Personal Computing Environment. *Communications of the ACM*, 29(3):184–201.

[Mullender, 1985] Mullender, S. (1985). *Principles of Distributed Operating System Design*. PhD thesis, Vrije Universiteit te Amsterdam, Netherlands.

[Mullender, 1996] Mullender, S. J. (1996). Me and my Smart Card. Invited talk, CARDIS 96.

[Mullender et al., 1995] Mullender, S. J., Corsini, P., and Hartvigsen, G. (1995). Moby Dick — The Mobile Digital Companion. LTR 20422, Annex I - Project Programme. Also available at URL: `http://wwwspa.cs.utwente.nl/~havinga/pp.html`.

[National Bureau of Standards, 1978] National Bureau of Standards (1978). Data Encryption Standard. NBS FIPS PUB. 46-1, U.S. Department of Commerce.

[National Bureau of Standards, 1980] National Bureau of Standards (1980). DES Modes of Operation. NBS FIPS PUB. 81, U.S. Department of Commerce.

[Needham, 1989] Needham, R. M. (1989). Names. In Mullender, S. J., editor, *Distributed Systems*, chapter 5, pages 89–102. ACM Press.

[Needham, 1993] Needham, R. M. (1993). Cryptography and Secure Channels. In Mullender, S. J., editor, *Distributed Systems*, chapter 20, pages 531–542. ACM Press, second edition.

[Needham, 1994] Needham, R. M. (1994). Denial of Service: An Example. *Communications of the ACM*, 37(11):42–46.

[Needham and Schroeder, 1978] Needham, R. M. and Schroeder, M. D. (1978). Using Encryption for Authentication in Large Networks of Computers. *Communications of the ACM*, 21(12):993–998.

[Neumann, 1995] Neumann, P. G. (1995). *Computer-Related Risks*. ACM Press.

[Newman and Lamming, 1995] Newman, W. N. and Lamming, M. G. (1995). *Interactive System DESIGN*. Addison-Wesley.

[Norman, 1983] Norman, D. A. (1983). Some Observations on Mental Models. In Gentner, D. and Stevens, A., editors, *Mental Models*, pages 7–14. Lawrence Erlbaum Associates, Hillsdale, NJ. Reprinted in R. M. Baecker & W. A. S. Buxton (Eds.), (1987). Readings in Human-Computer Interaction. Los Altos, CA, Morgan Kaufmann.

[Nyanchama and Osborne, 1993] Nyanchama, M. and Osborne, S. (1993). Role-Based Security: Pros, Cons, and Some Research Directions. *ACM Security, Audit and Control Review*, 11(2):11–17.

[Palmer, 1992] Palmer, E. R. (1992). An Introduction to Citadel - A Secure Crypto Coprocessor for Workstations. Technical report, IBM Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598.

[Postel, 1981a] Postel, J. B. (1981a). Internet Protocol. RFC 791.

[Postel, 1981b] Postel, J. B. (1981b). Transmission Control Protocol. RFC 793.

[Ramo, 1996] Ramo, J. C. (1996). Battle for the Future. *TIME Magazine*, 148(13).

[Rescorla and Schiffman, 1995] Rescorla, E. and Schiffman, A. (1995). The Secure HyperText Transfer Protocol. Internet draft (`http://www.eit.com/creations/s-http/-draft-ietf--wts-shttp-00.txt`). Expires 1/96.

[Reynolds and Postel, 1994] Reynolds, J. K. and Postel, J. B. (1994). Assigned Numbers. RFC 1700.

[Rivest, 1991] Rivest, R. (1991). The MD4 Message Digest Algorithm. In *Advances in Cryptology—Proceedings Crypto'90*, pages 303–311. Springer-Verlag.

[Rivest, 1992] Rivest, R. (1992). The MD5 Message-Digest Algorithm. RFC1321, Internet Network Working Group.

[Rivest, 1995] Rivest, R. (1995). The RC5 Encryption algorithm. Available via anonymous ftp: `ftp://theory.lcs.mit.edu/pub/rivest/rc5/rc5.ps`.

[Rivest et al., 1978] Rivest, R., Shamir, A., and Adleman, L. (1978). A Method for Obtaining Digital Signatures and Public-key Cryptosystems. *Communications of the ACM*, 21(2):120–126.

[Rivest, 1990] Rivest, R. L. (1990). Cryptography. In Leeuwen, J. V., editor, *Handbook of Theoretical Computer Science*, volume 1, chapter 13, pages 717–755. Elsevier.

[Rivest and Lampson, 1996] Rivest, R. L. and Lampson, B. (1996). SDSI—A Simple Distributed Security Infrastructure. `http://theory.lcs.mit.edu/~rivest/sdsi10.ps`. (Version 1.0).

[Roe, 1992] Roe, M. (1992). PASSWORD R2.5: Certification Authority Requirements. Technical report, Cambridge University Computer Laboratory.

[Rose, 1990] Rose, M. T. (1990). *The Open Book: A Practical Perspectice on OSI*. Prentice-Hall.

[Scheifler and Gettys, 1986] Scheifler, R. W. and Gettys, J. (1986). The X Window System. *ACM Transactions on Graphics*, 5(2):79–109.

[Schneier, 1996] Schneier, B. (1996). *Applied Cryptography*. John Wiley & Sonc, Inc., second edition.

[Simmons, 1992] Simmons, G. J., editor (1992). *Contemporary Cryptology, The Science of Information Integrity*. IEEE Press.

[Steiner et al., 1988] Steiner, J. G., Neumann, B. G., and Schiller, J. I. (1988). Kerberos: An Authentication System for Open Network Systems. In *Proceedings of the Winter 1988 Usenix Conference*, pages 191–201.

[Sterling, 1992] Sterling, B. (1992). *The Hacker Crackdown: Law and Disorder on the Electronic Frontier*. Bantam Books, New York.

[Sterne and Benson, 1995] Sterne, D. F. and Benson, G. S. (1995). The Controlled Application Set Paradigm for Trusted Systems. In *Proceedings of the 1995 National Information Systems Security conference*, pages 11–26.

[Syverson, 1994] Syverson, P. (1994). A Taxonomy of Replay Attacks. In *Proceedings of the Computer Security Foundations Workshop VII*, Franconia NH. IEEE CS Press.

[Syverson and Meadows, 1995] Syverson, P. and Meadows, C. (1995). Formal Requirements for Key Distribution Protocols. In Santis, A. D., editor, *Advances in Cryptology — EUROCRYPT '94*, volume 950 of *Lecture Notes in Computer Science*, pages 320–331. Springer-Verlag.

[Tanenbaum, 1988] Tanenbaum, A. S. (1988). *Computer Networks*. Prentice-Hall, second edition.

[Tanenbaum et al., 1990] Tanenbaum, A. S., van Renesse, R., van Stavern, H., Sharp, G. J., Mullender, S. J., Jansen, J., and van Rossum, G. (1990). Experience with the Amoeba Distributed Operating System. *Communications of the ACM*, 33(12):46–63.

[Thompson, 1984] Thompson, K. (1984). Reflections on Trusting Trust. *Communications of the ACM*, 27(8):761–763.

[TPEP, 1996] TPEP (1996). Common Criteria for Information Technology Security Evaluation, Part 1: Introduction and General Model, Version 1.0 (CCEB-96/011). Trusted Product Evaluation Program (TPEP).

[Voet, 1995] Voet, D. (1995). Design and Implementation of mIP: a Mobile Internet Protocol. Master's thesis, University of Twente, Enschede, Netherlands.

[Voydock and Kent, 1983] Voydock, V. L. and Kent, S. T. (1983). Security Mechanisms in High-Level Network Protocols. *ACM Computing Surveys*, 15(2):135–171.

[Weiser, 1991] Weiser, M. (1991). The Computer for the 21st Century. *Scientific American*, pages 66–75.

[Weiss, 1991] Weiss, J. (1991). A System Security Engineering Process. In *Proceedings of the 14th National Computer Security Conference.*

[Wheeler, 1993] Wheeler, D. J. (1993). A Bulk Data Encryption Algorithm. In *Proceedings of the 1993 Cambridge Protocols Workshop.*

[Wilkes and Needham, 1979] Wilkes, M. V. and Needham, R. M. (1979). *The Cambridge CAP Computer and its Operating System.* Operating and Programming System Series. Elsevier, North Holland.

[Williges, 1987] Williges, R. (1987). The use of models in human-computer interface design. *Ergonomics*, 30:491–502.

[Wobber et al., 1994] Wobber, E., Abadi, M., Burrows, M., and Lampson, B. (1994). Authentication in the Taos Operating System. *ACM Transactions on Computer Systems*, 12(1).

[Wright and Stevens, 1995] Wright, G. R. and Stevens, W. R. (1995). *TCP/IP Illustrated, Volume 2.* Addison-Wesley.

[Wulf et al., 1981] Wulf, W. A., Levin, R., and Harbison, S. P. (1981). *Hydra/C.mmp: an Experimental Computer System.* McGraw-Hill, New York.

[Yahalom et al., 1993] Yahalom, R., Klein, B., and Beth, T. (1993). Trust Relationships in Secure Systems—A Distributed Authentication Perspective. In *Proceedings of the 1993 IIII Symposium on Security and Privacy*, pages 150–164.

[Yee and Tygar, 1995] Yee, B. and Tygar, D. (1995). Secure Coprocessors in Electronic Commerce Applications. In *Proceedings of The First USENIX Workshop on Electronic Commerce*, New York.

[Yee, 1994] Yee, B. S. (1994). *Using Secure Coprocessors.* PhD thesis, Carnegie Mellon University.

[Zimmerman, 1994] Zimmerman, P. (1994). *PGP User's Guide, Revised for PGP Version 2.6.1.*