RC 16672 (#73914) 3/22/91
Computer Science 57 pages

Research Report

Introduction to the Citadel Architecture: Security in Physically Exposed Environments

Steve R. White, Steve H. Weingart,
William C. Arnold and Elaine R. Palmer

IBM Research Division
T. J. Watson Research Center
Yorktown Heights, NY 10598

KINDLY REPLACE THE REPORT
YOU HAVE WITH THIS REVISED
VERSION. THANK YOU,
DISTRIBUTION

IBM Research Division
Almaden • T.J. Watson • Tokyo • Zurich

Introduction to the Citadel Architecture: Security in Physically Exposed Environments

Version 1.4, May 30, 1991

Steve R. White
Steve H. Weingart
William C. Arnold
Elaine R. Palmer

Distributed Security Systems Group
IBM Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Abstract

We discuss the hardware and software architecture of a physically secure cryptographic coprocessor. The prototype hardware, named 'Citadel', was completed this year at IBM's Thomas J. Watson Research Center. We discuss its flexible architecture, which enables it to perform DES encryption and decryption of data at high speeds. We envision its use in three major application environments: 1) encryption and decryption of some or all network traffic (wire or fiber); 2) encryption and decryption of selected files or an entire disk; and 3) general purpose encryption and decription of passwords, user authentication transactions, resource requests, currency transactions, and other sensitive data from the main host processor.

Table of Contents

Preface

This document is an introduction to the Citadel architecture. It addresses the means by which systems can be protected from compromise due to physical attacks.

This work is part of a project in the Distributed Security Systems group in the Research Division. It draws upon architectural ideas from the IBM Transaction Security System [IBM91], the IBM Integrated Cryptographic Feature [IBM90b], and from previous work on secure processors in the Research Division.

This architecture is intended to be independent of the particular computing system, and computing environment, in which it is used. We will, however, use it as a basis for developing a prototype secure processor for an IBM PS/2 running under the OS/2 operating system.

Substantial changes from the previous version of the document are marked by bars to the left of the text, as on this paragraph. Minor typographical and punctation corrections are not marked.

We welcome your comments and suggestions. They should be sent to:

Steve R. White
(SRWHITE at YKTVMH) or srwhite@ibm.com

1.0 Introduction

The study of computer security has concentrated largely on the security of system software, such as the operating system. In traditional computer centers, with the computing facilities in a protected "glass house," this was very reasonable. Since users were prevented from getting physical access to the computing hardware, there was reasonable assurance that the system software would run as written. If the software protected users from accidental and intentional damage to the system, all was well.

Modern computing systems have grown well beyond the protective boundaries of the "glass house." They are now distributed throughout enterprises. Powerful workstations are in employees' offices, information terminals are in public places, and network traffic is carried on public phone lines and satellite links. The assumption that the physical elements of the computing system are not accessible is no longer valid.

The trend in computing is to distribute data and computational tasks across these large, distributed systems. Future systems will cooperatively process information on a global scale. As this trend continues, various aspects of the computing tasks will take place in a variety of physical environments, many of which are much less secure than the traditional "glass house."

Information is sent over networks which are subject to eavesdropping. It is stored on media like floppy disks, which can be carried off and examined or altered in complete privacy. Computing itself is done on systems in offices, in building lobbies, in shopping malls, and in automatic teller machines on street corners.

As more information assets are distributed to less secure environments, it becomes more important to protect these assets from attacks. This includes attacks against the exposed hardware of the system, as well as software attacks.

The purpose of this document is to introduce the Citadel architecture, which can help protect information assets in physically exposed distributed systems. "Security in Physically Exposed Distributed Systems" on page 2 discusses this class of security problem, the tools available to address them, and outlines how we propose to use these tools. "Models" on page 7 develops a number of models of systems, and of security in them. It shows how the tools developed in the previous section can be used to enhance the security of exposed systems. In particular, it introduces the idea of secure processors, an important element of the security of these systems. The architectural concepts we present for secure processors are intended to span the entire range of computing systems, from hand-held computers to mainframes, though different implementations may be quite different from each other. "Secure Processor Software Architecture" on page 19 discusses the software architecture of secure processors. "Secure Processor Hardware Architecture" on page 29 discusses their hardware architecture. "Physical Security" on page 42 introduces the notion of physical security of electronic systems in some detail. Finally, "Implementation Examples" on page 46 shows how secure processors could be implemented and used in several different systems.

2.0 Security in Physically Exposed Distributed Systems

Computer security consists of three aspects: secrecy, integrity, and availability. Secrecy of information ensures that access to it is restricted to those people (and processes) that are authorized to have it. Integrity of information and its processing ensures its correctness. Availability of information and services ensures their timely delivery to users.

Each aspect is important. Without secrecy of passwords, for instance, there is no way for a computing system to ensure the correct identification of users, and give them access to resources accordingly. Without integrity, users cannot be sure that the programs and data on which they depend remain correct and useful. Without availability, users cannot depend upon the system to provide them with the services they need.

2.1 Attacks Against Security

Traditional treatments of computer security concentrate on attacks that manipulate the logical inputs and outputs of a system. They are attacks against the operating system or the applications that it runs. We call these "logical" attacks, since they attack the logical integrity of the software running on the computing system.

There has also been concern about attacks on the physical hardware itself, but the study of these has not been nearly as systematic as has the study of logical security. We call attacks on the physical hardware "physical" attacks, to distinguish them from logical attacks.

2.1.1 Logical Attacks

There is a large body of work on the logical security of computing systems (see, for instance, [DENN82]). This work focuses primarily on the security of the operating system, and has resulted in evaluation criteria for secure operating systems [NCSC85]. If the operating system is secure, applications that run on it are properly isolated from each other. The idea is to limit the damage that could be done by a user introducing a harmful application to a system.

2.1.2 Physical Attacks

Traditionally, concern about the physical security of systems has centered on such things as fire, floods, electrical damage, and the like. Now that systems are small enough to be carried, theft of the system itself has been added to the list. These address the availability of system services, as well as the system itself as a capital asset.

Other aspects of physical security have become important, and will become more so as the trend towards distributed systems continues. Physical attacks can violate the secrecy and integrity of a system, as well as its availability [NESS87].

An attacker can remove a hard disk from a person's workstation, and install it on a different workstation. The data on the disk can then be examined, violating its secrecy. It can even be modified and returned to the original system, violating its integrity. Naturally, this is even easier to do with a floppy disk, which is designed to be both removable and portable between systems.

As smart cards become more widely used as repositories of information about individuals, the physical exposure of these important information assets grows. Under some circumstances, it may be possible to use hardware techniques to read information out of such cards, even when the cards are intended to keep that information secret. It may also be possible to alter information contained in the memory of these cards, such as the amount of credit in a cash card. This gives an attacker a way of forging the information.

In this document, we will discuss techniques for enhancing the security of exposed systems against physical attacks. We will show how this can enhance the security of the entire distributed system.

The issue of availability in the face of physical threats is both an old problem and a difficult one. It is often possible to deny access to an exposed element of a system simply by pounding it with a

hammer. This document does not address such availability problems. Instead, we will concentrate on the problems of secrecy and integrity in exposed systems.

2.1.3 Attacks on Exposed Systems

Logical and physical attacks against systems are interrelated. The correct operation of software depends crucially on the correct operation of the hardware that it runs on. Similarly, the secrecy and integrity of information that is stored or transmitted depends upon the properties of the storage or transmission hardware. Once the physical security of a system has been compromised, it may not be possible to ensure its logical security. Without logical security, it may be possible to compromise all of the system's information assets.

As an example, consider a workstation which stores the logon passwords of all its users in plaintext on a disk file. If an attacker steals this disk and examines the file, the attacker can then log on as any of the workstation's normal users. The logical security of the operating system is ineffective at preventing the attacker from gaining access to the users' resources. The same is true of a system which keeps a password in battery-backed RAM. Reading the memory with hardware may be somewhat more difficult than removing a hard disk, but it is well within the abilities of most engineers familiar with digital electronics.

Even if the important information assets (such as passwords) are not directly recoverable from the exposed part of the system, it may still be possible to use hardware techniques to accomplish this attack. Suppose a workstation is being used to interact with a mainframe, and the user logs on to the mainframe via the workstation. It would be possible for an attacker to insert a piece of hardware into the keyboard connector of a workstation that captures keystrokes in the logon sequence. When the user logs on, everything works as usual. But, in addition, the user's password is recorded in the attacker's hardware, and awaits later collection by the attacker. Again, this attack is well within the ability of most digital designers.

In traditional (logical) security discussions, a "Trojan Horse" is a program which is designed to do things that the user of the program did not intend for it to do. A logon Trojan Horse, for instance, presents the user with logon prompts, asks for the password, and so on. It may even perform the task of logging the user onto the correct system. But, in addition, it captures the user's password, and records it or sends it off to the attacker.

We refer to hardware devices such as the keystroke recorder as "physical Trojan Horses," since they consist of hardware that performs actions that their users did not intend. We refer to the more traditional ones as "logical Trojan Horses," to distinguish them.

Physical Trojan Horses present a threat very similar to that of their logical counterparts. Even in a system whose operating system is highly resistant to logical Trojan Horses, physical compromise of the system may accomplish the same ends.

2.2 Available Tools

At present, there are only three technologies which can address the above security problems.

2.2.1 Secure Operating Systems and Software

As previously stated, this has been the focus of a great deal of work on computer security. It is a vital component of overall security. Without the security of the operating system and other software, information assets will be increasingly at risk to remote users all over the world.

2.2.2 Cryptography

Modern cryptography permits data to be kept securely while being stored or transmitted. Cryptographic systems use a key, which is usually a small amount of information, to either encrypt or decrypt the information being protected. Keys can also be used to calculate cryptographic checksums of information, which can be used to certify that the information has not been changed since the checksum was last computed. Without knowing the keys, it can be extremely difficult to make a successful attack on the secrecy or integrity of cryptographically protected information.

Modern cryptosystems are too complex to be used by people without the aid of computers. Computers must be told the keys in order to use them. Once again, the logical security of the software that handles the keys may not be sufficient to ensure their security. In many systems, it is vital that the keys be kept physically secure as well.

2.2.3 *Environmental and Physical Security*

We use the term "environmental security" to refer to the class of techniques that ensures that unauthorized people do not have physical access to a system. This includes putting the system in a locked room, putting motion sensors and alarms in the room, monitoring the system with guards, and so on. These techniques are very well developed, and we will not discuss their details in this document [IBM72].

We use the term "physical security" to refer to the class of techniques that ensures that unauthorized people are unable to violate the secrecy or integrity of a system, even if they can get physical access to it. This is a relatively new field, and we will address it in some detail.

Environmental and physical security have similar goals and, as we will see, can be used to complement each other.

2.3 Using the Tools

The three tools discussed in the previous section can be combined to create effective means of protecting information assets from physical attack.

2.3.1 *Protecting Information and Processes*

The basic ideas of the Citadel architecture are very simple. Use cryptography to protect information assets when they are stored in, or transmitted through, exposed environments. When information must appear in plaintext (i.e. unencrypted), use physical security technology to make up for any lack in environmental security.

Information that resides passively on storage media, or on transmission media, need not appear in plaintext. In principle, all information assets could be encrypted with the strongest available cryptosystem whenever they are on storage or transmission media.

The only time that information *must* appear in plaintext is when an operation is being performed on it which depends on the object's information content. When a page of text is displayed to the user, for instance, it must appear in plaintext to be understood by the user. When a list is being sorted or a set of numbers is being summed, it must appear in plaintext so that the values can be used in the operation. When a program is being executed, its instructions must appear in plaintext in the processor in order to be executed. When valuable information does appear in plaintext, it must be protected by environmental or physical security.

2.3.2 *Secure Processors*

There must be a place where operations on plaintext objects can be performed, secure against physical attacks [KENT80]. We call elements of a computing system which provide such places "secure processors" [WHIT87, IBM90a].

In a distributed system, some processing elements will be in exposed environments. Their environmental security will not be sufficient to ensure plaintext objects against attack. So, secure processors must have sufficient physical security to make up for this lack in environmental security.

2.3.3 *Architectural Approaches*

Secure processors must be able to communicate with parts of the system outside of their secure boundaries. To do so securely, they must be able to encrypt at least some of their communications. This raises the important issue of overall system performance.

While, in principle, virtually all communication between the secure processor and the rest of the system could be encrypted, this may not be the best choice. Cryptography is very computationally

intensive. Requiring its use all the time can reduce the effective bandwidth of communications and storage channels.

There are two approaches to reducing this problem. First, it is not necessary to encrypt every object in every environment. Some objects may have such little value, and some environments may have sufficient security, that encryption is not required. In these cases, we can reduce the demand on the cryptographic resources by reducing the set of objects that must be encrypted.

The second approach exploits the trade-off between computational complexity and encryption strength. There is a variety of encryption systems, and they have a variety of strengths against various attacks. Given a certain expenditure of computational resource, it is possible to attain a higher cryptographic performance by using a less secure cryptosystem. This may be an acceptable approach when the value of an information asset is relatively low, or when the environmental/physical security of its location is relatively high.

The following sections examine several alternatives which make use of these trade-offs.

2.3.3.1 Data Link Encryption

The idea behind data link encryption comes from network security. In this scheme, all data objects that are to be sent on a given communications medium are encrypted with a given cryptosystem. When they reach the other end of the communications medium, they are decrypted. The term "data link" refers to the logical connection between the two ends of the communications medium in the OSI model [ZIMM80].

This technique offers the advantage of simplicity of implementation. Cryptographic facilities can simply be associated with each link. It is not necessary for any higher layer in the OSI architecture to be concerned with cryptography. It is transparent to the applications and operating system.

It has disadvantages as well. Since the encryption is independent of the object being encrypted, it cannot depend upon how valuable the object is. Since it is independent of the physical communications medium, it cannot depend upon the environmental/physical security of the actual communications path. A single cryptosystem must be sufficient for all possible system objects sent along all possible communications paths. In order to protect the most valuable objects along the least secure paths, a rather strong (and hence computationally intensive) cryptosystem must be chosen. Thus, a greater fraction of system resources is dedicated to cryptography than is necessary.

When used as a security policy for a secure processor, data link encryption has additional disadvantages. If a single cryptosystem is chosen to encrypt all objects as they leave the protection of the secure processor, then no object may leave the secure processor in plaintext. So, for instance, no object that was ever inside of a secure processor can be displayed as plaintext for a user. None can be printed out. None can serve as programs to be executed by non-secure processors. This is clearly a significant limitation.

2.3.3.2 End-to-End Encryption

End-to-end encryption arises from network security as well. In this scheme, it is the responsibility of each individual application to appropriately encrypt each object which it uses. The term "end-to-end" refers to the fact that objects are encrypted from one end of a logical transaction to another, regardless of the systems or communications media through which they travel. This scheme has the advantage of independence from the details of communications. It also has the advantage of being able to select a cryptosystem appropriate to the value of each object that it uses. It need not use an overly strong (and hence computationally expensive) cryptosystem for objects of low value.

It also has several disadvantages. Each application is burdened with the task of keeping track of the encryption needs of its objects. In this model, an application cannot be aware of where the objects will reside. As a result, the application must choose an encryption method which is sufficiently strong to ensure security in the least secure environment possible. It cannot choose a method which is less computationally demanding, even if the object will always reside in a relatively secure location.

Each application is also burdened with enforcing any global policy of cryptographic security that may exist. When such global policy decisions end up being enforced piecemeal by every application, it is traditional to absorb the decisions into the operating system, where they can be made more uniformly. This is the approach that we take in the next section.

2.3.3.3 Object-Oriented Encryption

We propose a new technique for handling the encryption of system objects in a uniform way throughout a distributed system. We call this technique "object-oriented encryption" because the security properties are associated with the objects themselves.

The idea is to classify objects according to their value or sensitivity, much as is done for mandatory access control policies [NCSC85]. A security policy is then formulated that dictates the environmental, physical, and cryptographic security requirements of system elements on which each class of objects can reside. The environmental security of a "glass house" could be sufficient, for instance, for a class of high-value objects to be kept or used there in plaintext. In workstations in open offices, these objects might require physical or cryptographic security to make up for the lack of environmental security. When transmitted on a public switched network, neither physical nor environmental security can be assured, so the objects may need to be encrypted.

Unlike the case in data link encryption, objects need not be encrypted beyond the degree required for their protection. Unlike the case in end-to-end encryption, applications need not be burdened with the responsibility of enforcing the security policy. By associating the security policy with the objects themselves, it becomes easier to enforce a uniform security policy throughout a distributed system.

A corresponding disadvantage of object-oriented encryption is the need to maintain databases of the environmental security characteristics of various system elements. When system elements are moved from one environment to another, these tables must be updated to reflect any change in environmental security.

3.0 Models

In this section, we develop several models that help us understand secure processors and the part they can play in the security of distributed systems. We develop a model of how secure processors interact with each other. This model shows how they can offer useful services to less secure parts of the distributed system. We show models of single systems and distributed systems, and show how secure processors fit into both. Finally, we give examples of specific roles that secure processors can play.

3.1 Models of Secure Processors

This section explores two models of how secure processors interact with other elements of a system.

3.1.1 Reference Monitor Model

In order to enforce a security policy in a system, it is necessary to control the ability of "subjects" to perform various actions on "objects." "Subjects" includes both users and processes. "Objects" includes files, memory areas, communications resources, and so on. The actions may include reading, writing, appending, and transmitting on. It must be possible, for instance, to ensure that a particular user has read access to a file, but not write access.

A "reference monitor" is an entity within a secure system that mediates all access by system subjects to system objects [ANDE72]. No user or process can do anything to any system object without going through the reference monitor. The reference monitor itself cannot be modified by any user.

The concept of reference monitors originated from the study of secure operating systems. As such, it is necessary to ensure that there are no *logical* attacks against the reference monitor's implementation as a security kernel [PIET87]. We can see that it is also crucial to ensure that there are no *physical* attacks on the security kernel either. If its physical security could be compromised, it may be possible to compromise its logical security as a result.

For these reasons, it is desirable to use secure processors to implement reference monitors. Objects can be kept secure, even when they are outside of the secure processors themselves, by cryptographic methods. If the objects are encrypted, and only secure processors know the decryption keys, then the objects are safe from attacks on their secrecy. If message authentication codes (MACs) or manipulation detection codes (MDCs) [JUEN85, IBM90a] are used, and if the secure processor keeps the MACs and MDCs for reference, then the objects are safe from attacks on their integrity.

In order to implement a true reference monitor, the secure processors must mediate *all* access between *all* subjects and *all* objects. If this mediation is done indirectly, through channels that do not have the appropriate degree of environmental, physical, or cryptographic security, a good deal of the benefit of using secure processors is lost. In particular, this means that all user inputs and outputs to the reference monitor must travel over a path that is environmentally, physically, or cryptographically secure.

The term "trusted path" is used in the security literature to refer to a guaranteed means for a user to communicate directly with the security kernel of the system [NCSC85]. We use the terms "physically trusted path" to refer to a communications channel between a user and a secure processor that is environmentally, physically, or cryptographically secure. We refer to the more traditional notion as a "logically trusted path," to distinguish it.

Thus, two things must be true for the secure processors to implement a true reference monitor. First, they must directly control all system subjects and objects. Second, there must be a physically and logically trusted path from the users to the security kernels within the secure processors.

It may also be useful to implement a reference monitor within secure processors which mediates all access to a certain set of objects. Cryptographic keys are an obvious candidate, since they often control valuable collections of information. It may be possible for non-secure subjects to use the cryptographic keys in certain, limited ways, such as generating MACs. But the ability to create or

change keys may be restricted to secure subjects. Access control records are another good candidate, since compromising their integrity compromises the security policy of the system. Non-secure subjects may be able to make use of access control decisions, but may not be able to alter the records themselves. Secure processors may be used as watchdog processors, to monitor events outside of themselves. This is similar to auditing external events, and will be discussed in a subsequent section.

User authentication can be done by the secure processors. In this case, it is necessary to have a logically and physically trusted path from the user to the security kernel within the secure processors. Any sensitive function performed by the system under user control must have similar guarantees. The system administrator, for instance, must have a trusted path to the secure part of the system. This may make use of physical security, in the form of tamper-resistant cables to I/O devices. Or, it may make use of environmental security, in the form of secure rooms in which system elements are located.

An auditing subsystem can operate in conjunction with a reference monitor to audit all actions performed by the security kernel. Such an audit subsystem can have great confidence in the accuracy and completeness of its records, in spite of physical attacks.

In general, the reference monitor model of secure processors provides a way for the secure part of the system to ensure the proper control of its subjects and objects. We will use the reference monitor model as the primary way for the secure parts of the system to interact with each other.

3.1.2 Server Model

A "server" is an element of a system that provides some set of services to other parts of the system. The idea of servers came from the study of distributed systems, in which some systems were chosen to provide more global services to the other systems. A file server, for instance, maintains a repository of files for many other systems. When a system needs a file, it makes a request to the file server, and the file server sends the file back.

Servers permit their services to be centralized, which makes access to these services easier to administer and control. Access to the objects controlled by servers is through a well-defined service interface. Servers can therefore mediate access to the objects they control, though not as completely as can a reference monitor.

There are several types of servers which are recognized as requiring security against physical attacks. Authentication servers [STEI88] require identifying information from users (userid and password, for instance), and then give users the ability to access certain system resources. They perform a function similar to logging the user on to a distributed system. If an authentication server were compromised, it could potentially compromise the entire system's security policy.

These sensitive servers are generally kept in "glass houses," or locked in closets. That is, they are kept secure by means of environmental security. An alternative to this is to use a secure processor as a server. By making up for a lack in environmental security with physical security, secure processors as serves permit critical services to be offered in a larger variety of environments. It would be possible to implement an authentication server on a secure processor that resides inside of a workstation, for instance. Even though the workstation is in a relatively insecure environment (the office), the authentication server may still be sufficiently secure. The workstation then has rapid, local access to this critical service, without sacrificing security.

In general, secure processors as servers provide a means for less secure parts of the system to have better access to critical system services. We will use the server model of secure processors as the primary way for the less secure parts of the system to interact with the more secure parts.

3.2 Models of Single Systems

We develop models of various systems, to understand the role that secure processors play in them. The first system that we will look it is that of a single machine. In this case, all of the hardware components of the machine are typically within a single enclosure, or in a single room.

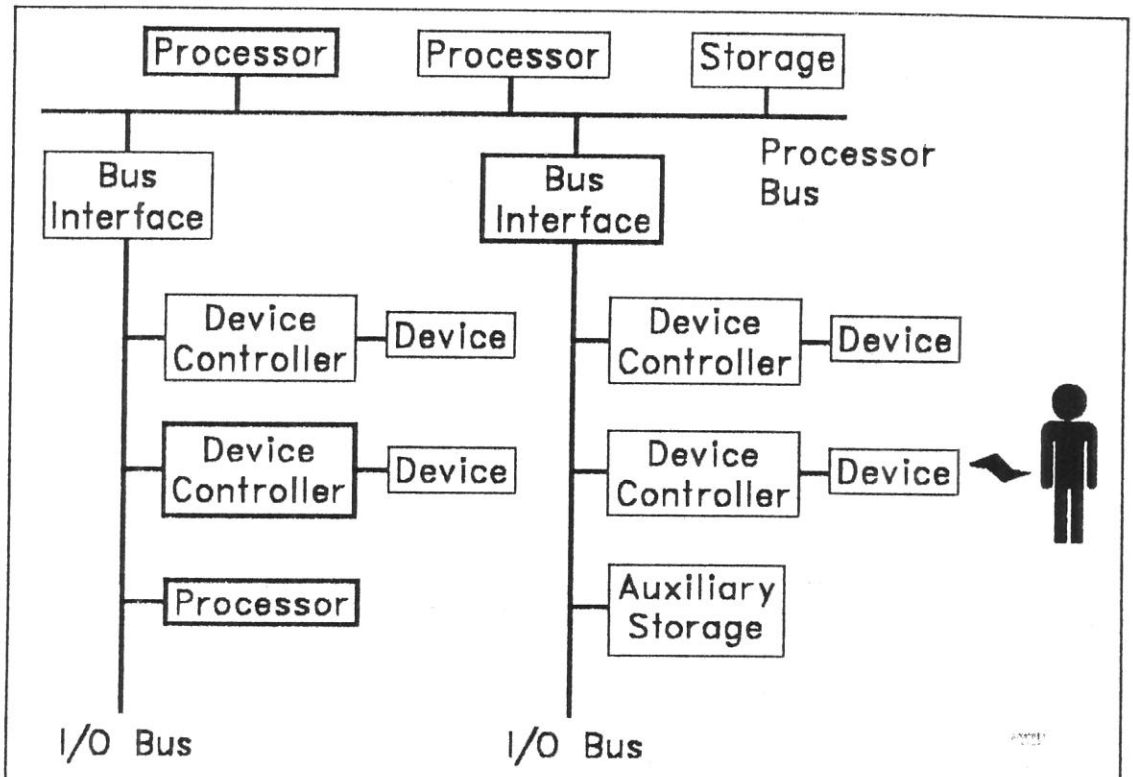

Figure 1. Model of a Generic Single System

3.2.1 *A Generic Single System*

Figure 1 on page 9 shows a model of a generic single system. This model is not intended to represent any particular machine, nor to include every element of every machine architecture. Rather, it is intended to show many of the major features of machines, and to be useful in thinking about the role of secure processors in them.

In the model, the main processor(s) communicate with primary storage on a processor bus. The processor bus is usually a high-speed bus, so that the processors can run at full speed, independent of the activity on the input/output busses.

The processor bus is connected to the input/output busses by bus interface units. The bus interface units handle requests from the processor for I/O services, and vice versa.

Various system elements are connected to the input/output busses. A device, such as a disk drive or a keyboard, is attached to a device controller, which is in turn attached to the input/output bus. The device controller may be an intelligent device, performing very complex actions with the device. It may be programmable, in the sense that it is possible to load programs onto it from external sources. It is also possible to have auxiliary storage and additional processors connected to the input/output busses.

Users do not usually interact directly with all of the elements of the system. Rather, they interact with some of the devices, which provide their only means of accessing the rest of the system.

3.2.2 *Secure Processors as Elements of a Generic Single System*

In principle, it is possible to make the entire system into a secure processor by putting it into a physically secure enclosure. If the system is small enough (e.g. a smart card), that may be a feasible alternative. If the system is large (e.g. a mainframe), it may not be.

When the entire system cannot be made into a secure processor, it may still be feasible to make certain elements of the system secure. The boxes with the heavier outline in Figure 1 indicate system elements that could be secure processors.

The primary processors of the system could be secure processors. They could act as reference monitors for the entire system, as outlined previously.

The bus interfaces could be made secure. They may be well placed to handle cryptographic traffic between the busses. Similarly, a device controller could provide pipelined cryptographic services to its devices. An auxiliary processor on an input/output bus could act as a secure server to the rest of the system.

These alternatives will be discussed in more detail later.

Another interesting alternative to consider is when a secure processor is included within the enclosure of the larger system. The enclosure of the larger system may act as a physical security system for the whole system. Its covers may be locked, and difficult to penetrate without a key or special tool. In this case, the entire system is moderately well protected from physical attack. It can safely process plaintext objects of a certain value, even in a less secure environment.

If the secure processor has an even higher degree of physical security than the overall system enclosure, it can safely process plaintext objects of a higher value. It can act as a reference monitor for subjects and objects that it controls. It can act as a secure server to the rest of the system. Its enhanced physical security broadens the tasks that can be safely assigned to the system.

3.2.3 Relation to Distributed Systems

When a single system is viewed as a collection of system elements, it has many of the characteristics of a distributed system. The system elements may consist of processors or devices of various sorts. Some may be intelligent. Some may be externally programmable.

Although the elements of a single system are usually in a single room, they do not have identical physical security characteristics. Hard disks in workstations are often easy to remove, which makes storage of valuable information assets on them risky. Information stored in battery-backed CMOS storage, on the other hand, can be more difficult to read or modify by physical means.

We now turn to a model of distributed systems in general, and will regard single systems as a special case.

3.3 Models of Distributed Systems

In a distributed system, the various system elements may be in a single location, or they may be in many different locations. Distributed systems may span a room, a building, a city, or the world.

3.3.1 Generic Distributed System Model

Figure 2 on page 11 shows a simple model of a distributed system. The system elements represented by boxes may be processing and/or storage elements. The elements may be intelligent, in the sense that they can perform very complex actions. They may be programmable, in the sense that they can load external programs. Any given element is assumed to reside in a location with homogeneous environmental security, such as within a single room. The elements may have differing degrees of physical security.

The system elements are connected via communications paths. These may be fixed communications paths like busses within a single machine. They may be more flexible communications paths like local area networks. Or they may be more abstract paths, such as those through a switched satellite network.

Users interact with some of the system elements, but not necessarily with all of them. There may be elements, such as servers or monitors, which do not have any direct interaction with users.

Figure 3 on page 12 shows an example of such a distributed system. Users interact with workstations, which communicate via a local area network. Also attached to the network is a file server, which may be locked in a closet. Users do not usually interact with the server directly.

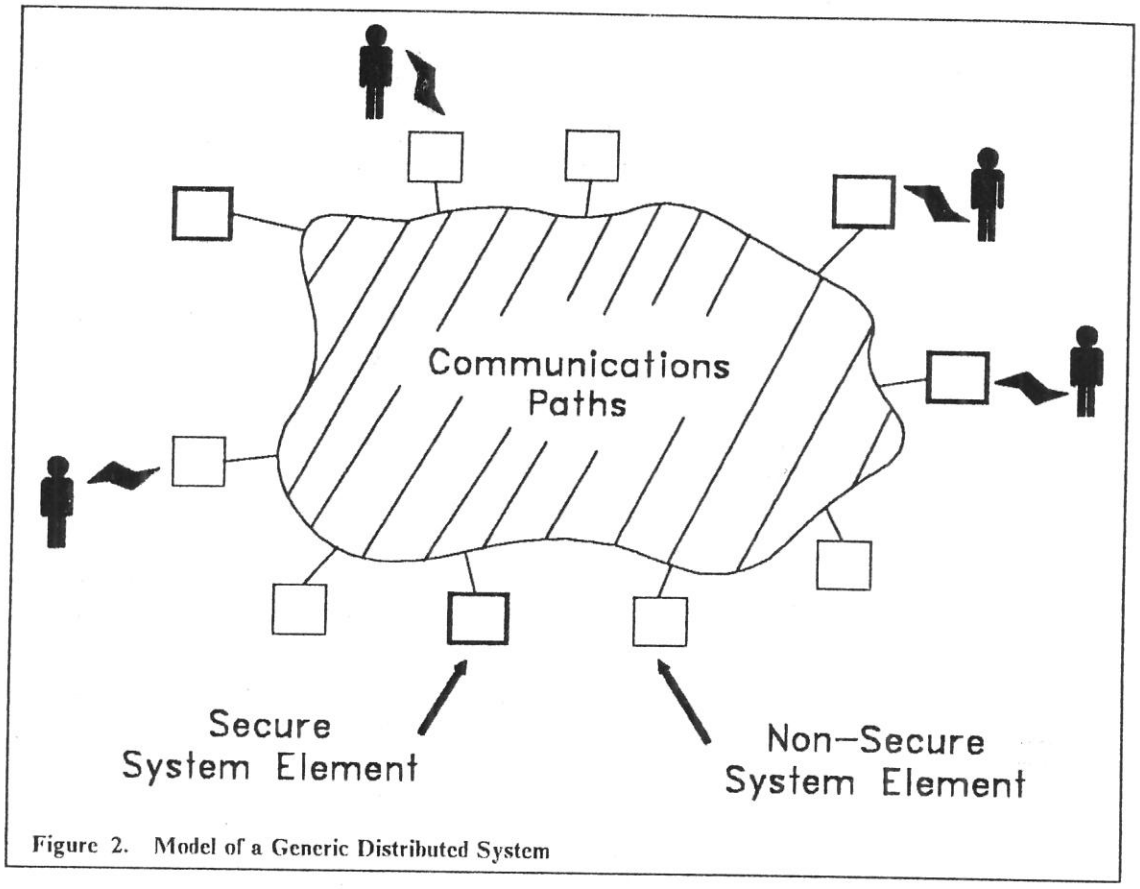

Figure 2. Model of a Generic Distributed System

Figure 1 on page 9 shows another example which fits our model of a generic distributed system. Users interact directly with some of the I/O devices, but not with the other elements of the system. The elements communicate along fixed paths formed by the system busses.

3.3.2 A Simplified Security Model

We can use this generic model of systems to study the effect of environmental and physical security on the security of the system.

3.3.2.1 The Model

Consider just two classes of environmental/physical security: trusted and untrusted. "Physically trusted" means that the physical and environmental security is sufficiently high that the resource in question is considered to be physically uncompromisable. "Physically untrusted" means that the physical or environmental security is not very high. This implies that it is possible for the resource in question to be compromised physically. This is illustrated in Figure 2 by some of the system elements being environmentally or physically secure (indicated by heavy boxes), and some not being secure in this sense (indicated by lighter boxes).

Note that there is a difference between "can be compromised" and "has, in fact, been compromised." Untrusted elements can be compromised, but have not necessarily been compromised yet. This is the same distinction as is drawn in the Orange Book [NCSC85] between "trusted" and "untrusted" computing bases.

It would do little good for the secure system elements in Figure 1 on page 9 to be environmentally/physically secure, but not to be logically secure as well. If that were the case, they could be compromised logically, without requiring physical access. We assume that secure system elements cannot be compromised either physically or logically.

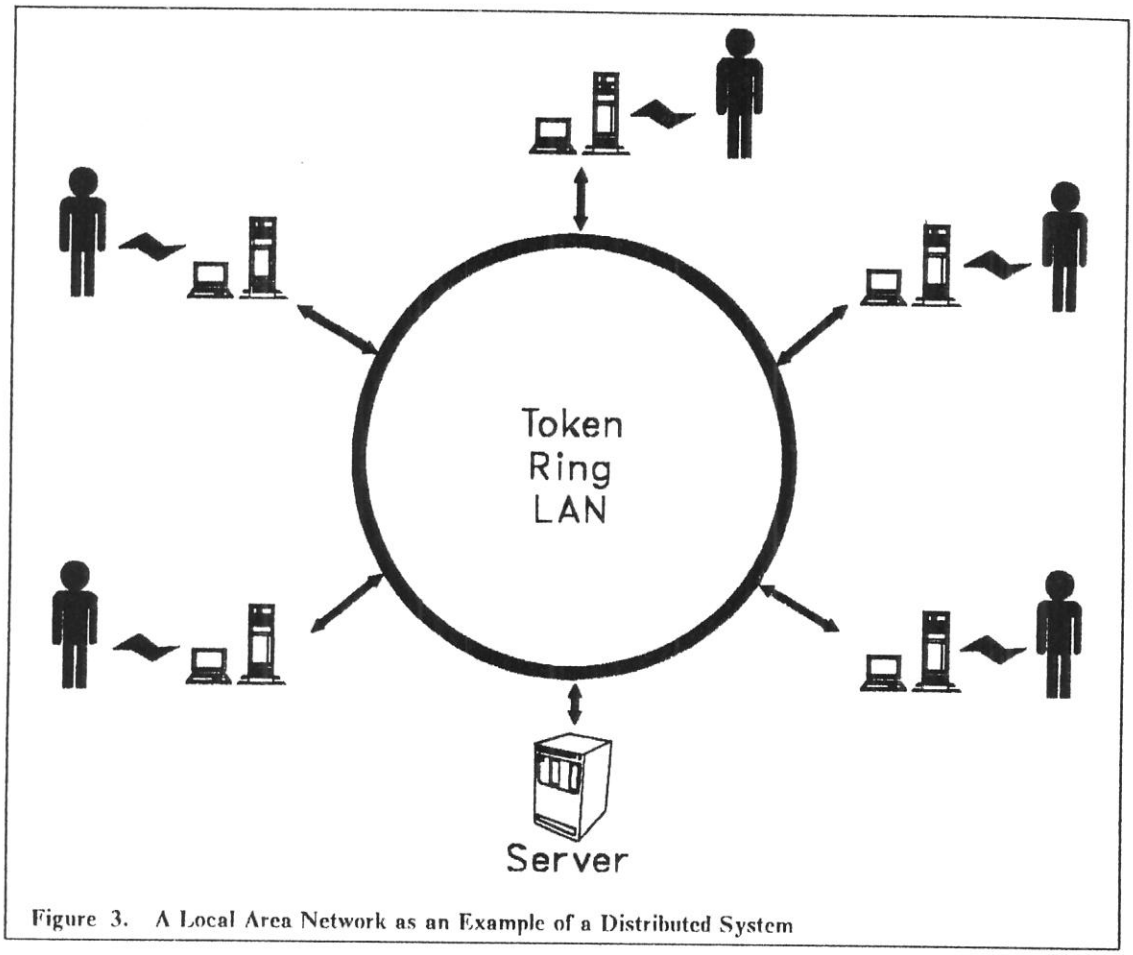

Figure 3. A Local Area Network as an Example of a Distributed System

In this simplified model, we will classify information assets as either "low value" or "high value." "High value" assets must either be made secure by environmental and physical security means, or they must be encrypted. In this section, we assume that there is just one cryptosystem, and that it is secure enough to protect the most valuable assets in the least secure environments. This assumption will be relaxed in the next section.

All communications between trusted processing elements could, in principle, be made cryptographically secure. This would prevent any compromise of the communications media. It would also enable assets of any value to be communicated between trusted elements.

This may not necessarily be feasible. Consider a workstation that has a processor and a disk controller on a bus. It is probably not feasible to encrypt all bus traffic between these elements, even though it fits the model of system elements connected to a communications path. Rather, we rely on the physical and environmental security of the bus to protect it from compromise.

If all traffic between trusted elements is encrypted, or along physically trusted paths, there are essentially two communications networks. One is between the physically untrusted elements, and the other is between the physically trusted elements. "Gateways" between the two must understand that they are translating information from untrusted to trusted.

Another alternative is just to guarantee that "high value" assets are encrypted when they are placed into (or transmitted through) untrusted locations. It is not necessary to encrypt all communications between trusted elements.

3.3.2.2 Attacks

In this simplified model, an attack on the logical integrity of a system element can only succeed if the element is logically untrusted. Similarly, an attack on the physical integrity of an element can only succeed if the element is physically untrusted. As was discussed previously, however, hardware attacks can alter the state or software characteristics of an element. Physically compromising an element can lead to it becoming logically untrusted. As a result, physical compromise can lead to logical compromise as well.

It may not be possible for a user to know that a particular element has been physically compromised except by direct inspection. Suppose that a physical attack leads to the introduction of a physical Trojan Horse in a system element. This Trojan Horse might know all of the secret identifying information of the processing element, such as encryption and decryption keys. This information is the only thing that allows a remote element of the system to verify the identity of this element. If the Trojan Horse has this information, it can "spoof" any remote challenge of its identity, and convince the challenger that it is the unmodified element.

This leads to an interesting situation. Consider the case in which the system consists of workstations on a network. Assume that the workstation enclosures themselves will not stop a physical attack, but will reveal that an attack has been made. The covers may be designed to fracture, for instance, when opened improperly. Suppose that each workstation contains a secure processor, within the workstation enclosure, and that this secure processor is physically trusted. If a user comes into the office in the morning and notices that her workstation's enclosure has been compromised, what network services can she trust? In this simplified model, she can trust none of them. The physically trusted path from the user to the uncompromised part of the system has been broken. The user can no longer trust the system to respond correctly to commands entered at the keyboard. The information on the display cannot be trusted either.

Does this mean that the rest of the network cannot trust the user's workstation? No. The secure processor is (by definition) still uncompromised. The other workstations on the network are still uncompromised. The other users have physically uncompromised paths to their secure processors. Those secure processors have a cryptographically trusted path to the secure processor on the compromised workstation.

The conclusion is that a local compromise of this sort may compromise the local user's use of the system. But the user's local system can still be useful to remote elements that have a trusted path to it.

Since the untrusted parts of the system may become compromised, the trusted parts of the system must regard them with some suspicion. Information that has appeared in the untrusted part of the system without cryptographic protection cannot be trusted.

How much damage could be done if some number of untrusted elements was compromised? At worst, information originating from the compromised part could contaminate information in other parts of the system, which could contaminate further parts, and so on. The compromise could be carried as far as the transitive closure of information flow in the system

This worst case may not always occur. Suppose that the attacker compromises an element, but does nothing else. Then there is no difference in the behavior of the system. No damage is done, except perhaps to the user's confidence in the compromised element.

An attacker can have more influence on the system by leaving Trojan Horses to gather information that can be used later. Suppose the attacker leaves a password gatherer on the compromised elements of the system. Some number of users log on to the system subsequently, and the attacker gets their passwords. The attacker then gains access to those parts of the system that could be accessed by those users. The damage that can be caused in this case is limited to what can be caused by those users working in collusion.

It is possible to design systems in which this damage is limited. A common example is to use a trusted (and secure) authentication server on a network, along with untrusted workstations. The compromise of a single workstation does not necessarily mean that the authentication server will be compromised. If no one but the attacker uses the workstation after it has been compromised,

the attacker does not discover any passwords. In this case, the attacker's access to the system may be severely restricted.

The consequence of compromising the physical elements of a system is precisely that of compromising the logical elements which they protect. Without environmental/physical security, logical security can be compromised. The specific design of the system determines the precise amount of damage that can be caused by a particular compromise.

3.3.3 A More General Security Model

The simplified security model in the previous section is too simple. In actual systems, parts of the system exist in a variety of environments, and have differing levels of physical security. In many cases, they also have differing degrees of logical security. Similarly, information assets span a range of values. Some are trivial, some are critical, and some are in between. This can be taken into account by generalizing the previous, simplified model.

3.3.3.1 The Model

In this model, we expand the two levels of physical trust in the previous model to many levels. In the physical security evaluation system in a companion document [WEIN88], we use five levels of environmental/physical security. Other schemes are also possible.

"Trusted" does not mean "uncompromisable" in this model. Rather, it indicates the extent to which it is difficult to compromise. Elements are more trusted because they are regarded as requiring more expense, skill, risk, etc. to compromise.

Define levels of physical trust, from "untrusted," through "moderately trusted," to "highly trusted." Again, these come from a combination of environmental and physical security.

Similarly, define levels of trust for the various cryptographic systems that may be employed. Plaintext may be "untrusted." Simple scrambling may be "somewhat trusted." DES encryption may be "moderately trusted." Triple encryption with DES may be "highly trusted."

More highly valued information assets that must be put into less secure parts of the system must be encrypted. The less the trust in the physical security of that part of the system, the greater the trust in the cryptosystem must be. Communication from less trusted parts to more trusted parts must be through "gateways" that understand this trust relationship.

Parts of the system that are more physically trusted should also be more logically trusted. Otherwise, there is an imbalance between the physical and logical security of the system. This could result in a successful attack on the operating system, say, even though the system's physical enclosure is quite secure.

3.3.3.2 Attacks

Attacks on systems in this model are a direct generalization of attacks on the simplified model. The system has been divided into separate "virtual networks" that have different levels of trustworthiness. Compromising a highly trusted element has the potential to compromise its virtual network, and those that are less trusted than it. This is because less trusted parts rely on more trusted parts for correct information. If the more trusted parts are corrupted, the less trusted parts can become corrupted.

3.4 Roles of Secure Processors

In the models of systems that we have discussed, information assets may reside in any element of the system which can protect assets of at least their value. Highly secure processors can handle plaintext assets which are very valuable, but they can also handle less valuable assets. Thus, secure processors offer a way to increase the scope of assets that may be handled securely in various environments.

In this section, we look in detail at various specific roles that secure processors can play in a system. When interacting with each other, equally trusted parts of a system can play the role of reference monitors for the assets which they control.

We orient this discussion primarily towards the interaction of more trusted and less trusted parts of the system. This highlights the benefits that can be achieved by incorporating secure processors into less secure systems.

3.4.1 Crypto Server

A crypto server performs basic cryptographic functions for the client. A client can request it to encrypt or decrypt a file under a specified key, create or check a MAC or MDC, and so on. Cryptographic keys are among the system objects that are commonly considered very valuable, since they can give access to a much larger set of objects. They are typically protected, both logically and physically, so secure processors are a natural place to keep them.

As a crypto server, a secure processor can be very flexible. It may be possible to load different cryptosystems into a single secure processor. Additional cryptographic functions, as well as corrections to implementation errors, may be made easily. It is possible to add a completely new set of crypto functions. A crypto server that was initially designed to perform DES operations, for instance, could be upgraded to perform RSA operations as well.

Remote crypto servers on an untrusted network are of limited value. If the network is not trusted, a workstation should not request the server to decrypt a valuable system object, and return it in plaintext.

It could be that an individual workstation is more trusted than the network, because it is environmentally more secure. It may still be important to offer greater security to the cryptographic unit. In this case, a secure processor located within a workstation provides a good solution. The moderately trusted workstation can make a request of the highly trusted crypto server, without the results being transmitted on the untrusted network.

Secure processors are designed to deal with high-value system objects. They are also designed to offer cryptographic services within the Citadel architecture. In many implementations, it will make sense to incorporate high speed cryptographic hardware in secure processors. This makes them especially attractive as crypto servers, since they will perform the cryptographic functions much faster than other parts of the system could.

Even when cryptographic hardware is not available in the secure processor, there may still be performance advantages. If, for instance, the RSA cryptosystem is implemented in software in a secure co-processor, the system can off-load RSA calculations to the co-processor.

3.4.2 Authentication Server

Authentication servers verify the identity of users, and give them certain permissions within the system. They perform functions in distributed systems that are the analog of logon facilities in single systems.

Since they provide access to many system resources, authentication servers and their databases are high-value resources. Classically, these servers have been locked in a closet, kept in a glass house, or otherwise made environmentally secure. Running the authentication service in a secure processor allows this service to be offered in a wider variety of locations.

In a system with a distributed authentication server, a server local to the user's machine could cache authentication information relevant to that user. This can improve response time, and can decrease dependence on network availability.

3.4.3 File/Database Server

A file server keeps a repository of files. When a client asks for a particular file, the server makes sure the client is allowed to possess that file, and sends it out if so.

A database server is similar. It maintains a database, and responds to queries to view or update the database if the client is allowed to see that view or make that update.

A database server may make more complex access control decisions than would a file server. The file server typically bases its decision on the identity of the client and the name of the file. A database server may base its decision in addition on calculations done on the database itself. An

assistant teller in a bank, for instance, may not be allowed to perform certain queries on accounts that have over a million dollars in combined assets. A database server on a secure processor could sum the account's assets, and determine whether or not the query is allowed. This can be done without exposing any information into the less trusted part of the system.

In either case, a secure server can provide these services securely across physically insecure communications channels. Secrecy can be assured through encryption. Integrity can be assured by using MDCs. This assumes that the client has a local crypto facility that is secure enough to handle the keys in this transaction. This may require a secure processor as well.

A secure file/database server differs from a secure crypto server. The file/database server does not necessarily need to be told which keys to use in doing the encryption. It does not need to be told whether to encrypt, nor whether to do an MDC calculation. This information is associated with the file system or database system directly.

If object-oriented encryption is used as part of the Citadel architecture, then the correct crypto manipulations are built into the underlying system. In this case, every server is a secure server.

A server which is secure enough to handle high-value objects can also be used to handle objects of lower value. High-value and low-value files can be kept on the same storage device, and used on the same system. The low-value files may not need to be encrypted while on the storage medium, so the system's crypto resources are not burdened by them. A secure processor that is local to a user's workstation can act as a secure file or database server to the less trusted parts of the workstation.

Audit and access control on the movement of the files and database queries is strong. The secure processor can supply to the user only those views of the database that the user is authorized to see. This enforces strong control over the database as a whole.

3.4.4 Access Control Server

An access control server gives its clients access to certain system resources. In typical mainframe implementations, an access control *facility* returns "true" or "false" to the client program, telling it that the requested access is allowed or not. This is sufficient if the client is at least as trusted as the server. Objects used entirely in trusted parts of the system can be controlled quite strongly this way.

If the client is less trusted, an access control *server* must do more. Suppose a program in the less trusted part of the system requests an access control decision from the more trusted part. If the access is denied, the program may be free to perform the access anyway. Simply being told not to do it is insufficient protection.

Instead, an access control server must couple its decision about whether or not the access is allowed, with the capability to access the object. It may give a key to the requester. It may return a Kerberos access ticket [STEI88]. It may, in some cases, give the user the plaintext object itself (though this provides less protection of the object subsequently).

3.4.5 Audit Server

An audit server records information about transactions that it can observe directly, or about ones that it is requested to record. Audit trails can be a significant way of tracking down a system penetration or security violation. Auditing elements often contain sensitive, high-value information, and are good candidates for implementation in secure processors.

An audit server can securely audit all system actions performed by uncompromised parts of the system. For transactions that occur in parts of the system that are at least as trusted as the audit server, the audit server can have great confidence in the accuracy and completeness of its records.

It is useful to maintain an uncompromisable audit trail of events that can be audited, even if the events occur in the less trusted parts of the system. An attacker's entrance to a less trusted part of the system may be recorded by the audit system because that part of the system is still uncompromised. Even if the attacker subsequently compromises that part of the system, a secure audit server can prevent the attacker from altering the audit records to cover up the attack.

As a co-processor, an audit server local to a user's workstation may have significant performance advantages. If all auditable system events are audited, system performance can be significantly degraded by the time required for auditing. If this task is off-loaded to a secure co-processor, it may be possible to recover much of this performance.

3.4.6 Execution Server

An execution server loads and executes programs for its clients. It can perform the functions of any of the above servers, by loading and executing the server software. In this sense, the abilities of a secure execution server encompass the abilities of all other secure servers.

In addition to performing as one of the general types of servers discussed above, secure execution servers can be used for more specific functions. An element of the system may need a task performed in a more secure environment than its own. It can send this task to the execution server, and expose only the result into its less secure environment. The task that the execution server was asked to perform need not be generally useful to other elements of the system. It may not justify a separate server of its own. An execution server can handle many such special-purpose requests.

A business may be concerned about the integrity of the program that reconciles accounts receivable with accounts payable. This program may run only once every few weeks, so it doesn't justify a separate secure server by itself. It could be run when needed on a secure execution server.

The designer of an aircraft part may wish to perform an environmental stress test on the part. If the criteria of the test are sensitive, due to their competitive or national defense nature, the test could be done on a secure execution server.

In some cases, it is important for the server to ensure the integrity of the program before it is executed. This is generally a good idea, since the purpose of a secure execution server is to execute the requested program, not some altered version of it. It is especially important if the program will be given significant privileges in the secure processor, as would a program that performs administrative analysis and updates on the processors.

The integrity of the program can be ensured by using MACs or MDCs, and storing the values necessary to calculate them securely. Before control is transferred to the loaded program, the execution server checks the program's MAC or MDC. If the program does not correspond to one of the allowed values, the execution server will not run it.

In some cases, it is important for the program to ensure that the server on which the program executes is trusted. This is the case when a program is of high value, or controls high-value objects. It would be unacceptable for another execution server, which may not be secure, to "spoof" the client into letting it run the program.

This can be accomplished by encrypting the program, and giving its decryption key only to authorized, trusted execution servers. Without the key, the program cannot be exposed in plaintext. Any time the program is executing, it can "know" that it is executing on a trusted execution server.

The use of MDCs and decryption keys outlined above can be combined. This allows the program and the execution server to mutually assure each other that they are valid. It is analogous to the cryptographic protocols for mutual authentication [MEYE82]. The difference is that in this case, one of the parties (the program), may exist entirely on storage media, and may have no independent processing power of its own.

3.4.7 Assured Loading of Programmable Elements

Much of the preceding discussion of roles has focussed on secure processors as secure servers. By permitting them to load and execute external programs, they can be very flexible in these roles. It is possible to control which programs can be loaded into which servers, which helps ensure that they are performing their intended functions.

There remains the problem of guaranteeing the correct IPL behavior of a collection of secure processors, and the correct global behavior of the collection as new programs are loaded. This can be seen as follows.

Suppose that two elements run different parts of Version 1 of a software package. Version 2 of the package comes out, and we want to update both elements. But, the changes between the two versions prevent us from having a secure system if only one of the elements has been updated. They must be updated simultaneously to be safe.

The problem is more general than this. We have proposed a distributed system with many execution servers. In such systems, servers will periodically change the program(s) that they are running. How do we ensure that the proper global state of the collection of execution servers is maintained?

One way to do this is to designate a single element, which will control the loading of the other elements. Its purpose is to coordinate global state changes among the other elements. We will call the controlling element the "master," and will call the other elements the "slaves." We use these terms solely for discussing the global state problem. This is not meant to imply that these elements have a master-slave relationship for other system functions.

On power-up, the master must initialize itself securely. It must have access to storage media if it needs to be initialized with programs from outside of itself. Ensuring the correct power-up state of a secure processor is discussed in "Secure Processor Software Architecture" on page 19.

The slaves can do a very simple power-on initialization, and wait for commands from the master to load programs. The master can open a communications channel with each slave in turn, and send each slave the proper programs. The master must use a communications channel with sufficient environmental, physical, or crypto security to protect the programs being distributed.

After initialization, a slave can start the program it was given. The master may, at any time, command a slave to cease execution and await a new program. The slave could respond immediately, or could complete its current task before responding. It then signals the master that it is ready to have its program changed.

The preceding discussion assumes that having one or more slaves waiting for new programs is always an allowed global state. This makes it simple for the master to order a slave to stop what it is doing, and wait for new instructions. This seems like a straightforward property to ensure.

Assuring the global state of a collection of secure processors is not primarily an interaction between more and less trusted parts of the system. As a result, it is not primarily a server role. It is an interaction of the more trusted parts of the system with each other to ensure its own global state.

4.0 Secure Processor Software Architecture

This section describes a proposed architecture for the software components of the Citadel secure processor architecture. It includes reasoning supporting the architectural decisions made.

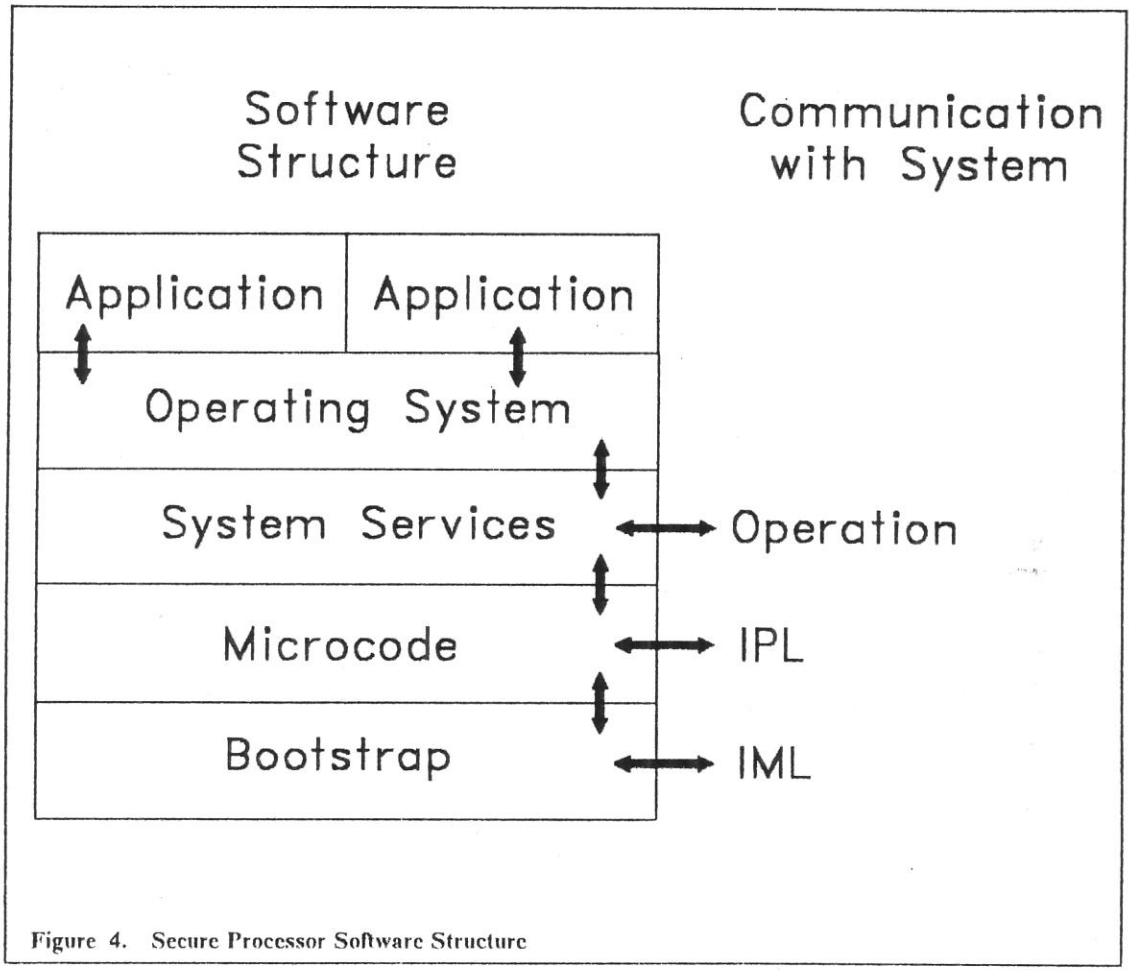

Figure 4. Secure Processor Software Structure

4.1 Introduction

The architecture is layered to permit configurability and field upgrade of the software. Figure 4 diagrams the layers. Each layer is responsible for loading any layer directly above it in the diagram. The bootstrap layer is never loaded, and there are no architected mechanisms through which applications can directly load other software. This layered software loading procedure is a commonplace method of reaching a desired software state in a system.

There are several common motivations for this sort of layered loading of software. A motivation for layering is that the different layers may be sufficiently distinct that they can be written by different organizations. Applications might not be written by the same organization that writes the microcode, for example. A motivation for layered loading is that it is a way of minimizing the size of persistent storage in a processor. The persistent storage may have special characteristics which increase its cost.

There is a particularly important motivation for layered loading of software, peculiar to the Citadel architecture. It is possible (and part of the Citadel architecture) to provide cryptographic assurances of the integrity and secrecy of every layer that is loaded on top of the microcode layer. Integrity

controls can provide strong assurance that code has not been illicitly modified. Secrecy controls provide indirect assurance to program objects of the identity of the environment in which they are executing. A program cannot execute unless it has been decrypted. If a program is normally stored in encrypted form and is now executing, then it must have been decrypted. Thus the program can implicitly "know" that it has been decrypted by some agent which possesses the correct decryption key.

The storage areas of a Citadel secure processor are the ROS (Read Only Storage), the persistent storage, the primary storage, and the secure data storage. The ROS is not alterable, and retains state through a system power cycle. It is installed in the factory. The persistent storage is any Read/Write memory that can maintain state through a system power cycle. The primary storage is Read/Write memory that is used by programs and data and does not necessarily maintain state through a system power cycle. The secure data storage is Read/Write memory that retains state through a system power cycle, like the persistent storage. But, unlike the persistent storage, it is used to hold sensitive and valuable information. For further details, see "Storage" on page 37

The bottommost layer, the bootstrap layer, enables the IML (Initial Microcode Load). It should reside in ROS. In the secure processor, it would typically load a layer of microcode into persistent storage. It would do little else, mainly because it cannot be upgraded in the field because the secure enclosure would have to be breached.

The system microcode layer provides I/O support, including support for the cryptographic hardware, and securely loads the system services layer. The system microcode layer would typically be present when the secure processor was powered up.

The system services layer implements services that would normally be part of an operating system kernel. Comprehensive cryptographic support is included, as is a security kernel used by the next layer, the operating system layer.

The operating system layer simulates an operating system for which development tools already exist. Typically it would simulate an operating system commonly used somewhere in the computing environment of which the secure processor is a part. This layer would expose an operating system interface to applications, but the interface would incompletely simulate the operating system; calls that could result in a security exposure would surely not be implemented, and other calls might not be implemented for other reasons. Some calls might not be implemented simply because they would be difficult to implement securely.

The application layer consists of the loaded programs that enable the secure processor to perform a role in the overall computing environment. Applications could act as server processes or perform other functions.

The architecture outlined in this chapter may be subsetted in some circumstances. For example, a smart card implementation would not normally load large parts of its software from outside the smart card during normal use. Instead, a smart card would normally keep the software resident at all times. A smart card implementation might also dispense with several of the layers. For example, the system services and operating system layer might be eliminated, and the system microcode layer could load applications directly.

4.2 Bootstrap Layer

4.2.1 Physical Placement of Bootstrap Layer

The bootstrap layer would typically be stored in ROS, so that it would always be present upon power on, including initial power on during the manufacturer's test phase. This layer cannot be updated without opening the secure enclosure. It can be considered an invariant part of the secure processor configuration.

4.2.2 *Purpose of Bootstrap Layer*

4.2.2.1 Purpose, Assumptions and Constraints

The bootstrap layer is intended to be a minimal amount of code that is given control on power up and enables the loading of the full complement of microcode. We recommend that this layer be minimal for several reasons. One good reason is that this layer cannot be updated in the field, so any security-related bugs found in this layer could necessitate withdrawal of units from the field. Another reason is that the manufacturer may not be the sole vendor of microcode for such a device. For example, some institutions may want to use their own microcode to implement special purpose and possibly proprietary systems.

4.2.2.2 Cryptographic Keys Associated with System Microcode Layer

The bootstrap layer loads the system microcode layer. There are cryptographic keys associated with the system microcode layer. These keys would typically be loaded along with the system microcode layer. They are used to assure the integrity of the system services layer that is subsequently loaded and the secrecy of the keys that are loaded with the system services layer. These keys are at the base of a recursive loading process that ensures the eventual integrity of the entire software state of the secure processor. Access to these keys should be carefully limited to only those individuals or agents with the need to know or possess them. In many circumstances, these critical keys could be machine generated and automatically installed, so that no human being needs to know them.

4.2.3 *Bootstrap Layer Control Flow*

4.2.3.1 Power On Case

The bootstrap code has at least two entry points. One is used when the secure processor is powered up. Minimal diagnostics are performed, and then a signature area (see "System Microcode Layer Signature" on page 22) is checked to determine whether or not there is a valid system microcode layer present. If there is, then the system microcode layer is given control. If there is not, then the secure processor signals an error condition and waits.

4.2.3.2 Initial Load or Update of System Microcode Layer

Another logical entry point is used if an external agent wants to update the system microcode layer. For example, assume that the secure processor manufacturer ships an update of the system microcode to a site. A trusted individual might be given the task of updating the system microcode layer and loading the keys associated with the system microcode layer. This individual might perform this update by connecting a secure processor directly to an isolated trusted workstation in an environmentally secure location.

There must be an uncompromisable mechanism that the agent can invoke that guarantees that this entry point is used; if this were not the case, then a subversive system microcode layer could prevent its own update. This mechanism could be implemented in a variety of ways. An example would be to have the POST (Power-On-Self-Test) code in the bootstrap layer check the state of a port which an external agent would use to request a microcode update. If the port signaled a request for a microcode update, then update code would be invoked.

The secure processor first verifies minimal hardware integrity. The bootstrap code must verify that the memory areas used by the system microcode layer function as designed. In our hardware architecture (see "Secure Processor Hardware Architecture" on page 29) these consist of the persistent storage, the secure data storage, and the portion of the primary storage designated for use by the system microcode layer.

Then all state information in the secure processor is erased.[1] This might be done by setting all read/write memory and registers, so that the secure processor is in a single known state.

After the bootstrap layer has done its minimal hardware integrity check and put the entire secure processor into a known state, it accepts the system microcode layer as input from outside the secure boundary. It then writes the system microcode layer into the persistent storage.

There is an obvious problem with this approach to loading the microcode layer; any external agent can load a new microcode layer in the secure processor. This is not a severe problem. Any cryptographic keys associated with the previous microcode layer are erased before the new microcode layer is loaded, so the new microcode layer cannot load the system services layer unless the correct keys are loaded. (See "Alternative Cryptographically Secure Approach" on page 23 for another approach)

4.2.3.3 System Microcode Layer Signature

The bootstrap needs to be able to decide whether or not a valid system microcode layer has been loaded, before it passes control to that layer. Because the secure processor's state information is erased before making modifications (including initial load) of the system microcode layer, a robust signature is not necessary except for reliability reasons. A microcode layer cannot load the system services layer unless the correct decryption keys are available. A simple signature embedded in the system microcode layer might be sufficient; an alternative stronger approach is to store a checksum of the system microcode layer (or CRC code) and verify the checksum before passing control to the system microcode layer. Similarly, a checksum could be performed on the part of the secure data storage area used by the system microcode layer.

The system microcode layer uses certain cryptographic keys to assure the integrity and secrecy of the system services layer. As noted in "Introduction" on page 19, the secrecy assurance is also indirect assurance to the system services layer of the correct identity of the system microcode layer. If the system services layer is stored in ciphertext and only decrypted after being loaded into the secure processor, and the system microcode layer is the only agent that knows the decryption key for the system services layer, then the system services layer implicitly "knows" that it has been decrypted by the system microcode layer.

These cryptographic keys might be loaded and installed in the secure data storage, concurrently with the loading of the system microcode layer. They might instead be loaded and installed after the bootstrap layer has determined that the system microcode layer is valid but before it is given control. Another alternative is to load and install them upon request by the system microcode layer itself.

4.2.4 Lack of Cryptographic Services, and Resulting Integrity Considerations

4.2.4.1 Reasons for Lack of Cryptographic Services

Cryptographic services are not available to the bootstrap layer. The same reasons for minimizing the size of the bootstrap layer (See "Purpose, Assumptions and Constraints" on page 21) apply to the exclusion of cryptographic services from the bootstrap layer. For example, not all customers will want to use the cryptographic algorithms that the vendor of secure processors supplies.

4.2.4.2 Implications of Lack of Cryptographic Services

As noted above, the security of the recursive loading process rests on the secrecy of keys loaded concurrently with the system microcode layer. (See "Cryptographic Keys Associated with System Microcode Layer" on page 21) It is very important that these keys not be compromised. There are no cryptographic services available to the bootstrap layer, so it is not possible to set up a cryptographically secured communication between the secure processor and the agent which wishes to load the system microcode layer and associated keys. The transaction should be end-to-end

[1] Properly, an analysis could be done to determine what subset of the state information would have to be erased to prevent a security breach. For simplicity of analysis, we recommend that all state information be erased. The microcode is updated infrequently, so the time spent doing the extra work should not be bothersome.

physically or environmentally secure. For example, the microcode might be loaded in an environmentally secured room, using a trusted workstation which does not have any network connections.

Additionally, note that the agent wishing to load the system microcode layer has no way of establishing the identity of the secure processor. The agent must trust that it is communicating with an uncompromised secure processor rather than with a Trojan Horse.

4.2.4.3 Alternative Cryptographically Secure Approach

An alternative, cryptographically secure approach to loading the system microcode layer is available. Associated with the system microcode layer are cryptographic keys. These keys could be used to set up a cryptographically secured channel between the bootstrap layer and the agent loading the microcode layer. The first time microcode and keys were loaded, they would have to be loaded insecurely; subsequent loads could be made secure. The secure processor would record that an initial microcode load had been done and enforce the cryptographic security of subsequent microcode loads.

There is a weakness with this approach. The agent securely loading the microcode layer must know cryptographic keys that were previously loaded. For example, if the manufacturer were to perform the first-time load of the microcode layer and keys, then an update of the microcode layer would either require the direct intervention of the manufacturer, or that the manufacturer share the keys with the customer. If the copies of the cryptographic keys external to the secure processor were somehow lost, the secure processor's microcode layer could no longer be updated.

Another weakness is that, as mentioned in "Reasons for Lack of Cryptographic Services" on page 22, the bootstrap layer cannot be updated without opening the secure enclosure. This means that the complexity of the bootstrap layer should be minimized, so that there is some confidence of its correctness. The cryptographic support necessary to support a secure microcode load is complex.

4.3 System Microcode Layer

4.3.1 Definition

Included in this layer is software support for I/O, including support for the cryptographic hardware (if any), and the clock (if any). This low-level support should only be exposed to the system services layer, through controlled interfaces. In many implementations, direct I/O to physical devices would be confined to this layer.

The second major part of the system microcode layer is its ability to load the system services layer securely.

4.3.2 Physical Placement

Typically, the system microcode layer would be loaded into some sort of rewritable persistent storage such as EEPROM. It could conceivably be loaded each time the secure processor power was cycled, or more often. This would probably not be optimal, because associated with this layer are sensitive cryptographic keys, and in the typical environment where a secure processor is used, these keys could not be securely loaded. (See "Reasons for Lack of Cryptographic Services" on page 22)

4.3.3 Availability of Lowest Level of Cryptographic Support

Cryptographic support is introduced in this layer. Typically, the support would consist of I/O support to the cryptographic hardware, or a similar interface to software cryptographic support. Support would typically be limited to a software interface to the cryptographic services directly supported by the cryptographic hardware or software.

As an example, consider a secure processor that has DES hardware. The system microcode layer would support software interfaces to various DES modes, such as code book mode, cipher block chaining mode, etc.

4.3.4 *Power On Self Test (POST)*

Some minimal testing is done at power on by the bootstrap layer. The rest of the power-on-self-testing should be done by this layer. In particular, hardware security mechanisms should be tested where possible, and the integrity of system memory should be thoroughly tested before loading any important software and/or data. The POST should be thorough, because the integrity of the security software rests on the integrity of the hardware base into which it is loaded.

4.3.5 *Cryptographic Keys Loaded with this Layer*

As is outlined in "Bootstrap Layer" on page 20, cryptographic keys are loaded concurrently with this layer. These keys are used to provide cryptographic assurances of the secrecy and integrity of the system services layer. The secrecy assurance is also an indirect assurance to the system services layer of the correct identity of the system microcode layer. (See "Introduction" on page 19)

The cryptographic support that this layer provides to the system services layer does not typically include support for manipulation detection codes (MDCs) or message authentication codes (MACs). Both can be used to assure integrity. If the cryptographic hardware did directly support such integrity verification functions, then they would be included in the cryptographic support that this layer provides to the system services layer. In the absence of hardware support for such functions, this layer would typically implement some form of MDC or MAC for its own use, using the available cryptographic services.

The primary reason for including the comprehensive cryptographic support in the system services layer rather than the system microcode layer is that the customer may not want the vendor supplying the system microcode layer to be the same as the vendor supplying the system services layer. A secondary reason is that comprehensive cryptographic support may require more space than is available for the system microcode layer. Some implementations may include the comprehensive cryptographic support in the system microcode layer.

4.3.5.1 Mutually Suspicious Exchange with System Microcode Layer

This layer can be loaded by any external agent. A secure processor loaded with subversive software could attempt to masquerade as a secure processor loaded with unsubverted software. The only way of ensuring that a secure processor has really been loaded is to challenge it from another (presumably) secure processor. The computation of the correct response should be impossible without possession of secret information, such as the keys that are loaded with the system microcode layer.

A correct response to the challenge is only an indirect indicator of the integrity of this layer. The bootstrap layer can ensure that any state information in the secure processor is erased before it permits new microcode and keys to be loaded, but the agent loading the system microcode layer has no guarantee that it is communicating with the bootstrap layer. A correct response to the challenge is evidence only that some other process than the challenging process was in possession of the secret information from which a response can be computed.

The guarantee of identity of the system microcode layer provided by a correctly completed challenge-response sequence is only as strong as the guarantee that an imposter does not possess the key necessary to compute a correct response. As noted in "Bootstrap Layer" on page 20, it is very important that the loading of the system microcode layer be end-to-end physically and/or environmentally secure.

4.3.6 *Communications with this Layer*

The system microcode layer provides software interfaces to the support for the secure processor's various I/O devices. In particular, it provides software interfaces to the cryptographic hardware. Typically, these interfaces would only be directly used by the system services layer, the layer loaded after this one.

Some support for communications with the secure processor also needs to be included in this layer. Enough communications support needs to be included so that the system microcode layer can load

the system services layer. Software interfaces to the support for the communications hardware should also be included, and these interfaces should be provided to the system services layer.

4.3.7 *System Microcode Layer Control Flow*

When this layer is given control by the bootstrap layer, the first action that it performs is the POST, as described in "Power On Self Test (POST)" on page 24. Minimally, this layer has to test the resources used by the system services layer which it loads. Sensibly, this layer should test the entire secure processor fairly thoroughly.

After the POST is completed, this layer determines whether or not it needs to load a system services layer. If there is a system services layer already present, in for example persistent memory, then a decision has to be made of whether or not to reload the system services layer. Implicitly, this is a decision of whether or not to trust the integrity of an already present system services layer.[2] If a previously loaded system services layer is trusted by default, organization policy should somehow ensure that an insecure system services layer is never loaded and subsequently incorrectly trusted. If the system services layer is already present and trusted, then this layer passes control to the system services layer. If there is not a trusted system services layer loaded in the secure processor, then one needs to be loaded.

The process of loading the system services layer securely is repeated analogously whenever software is loaded from outside the secure boundary. The system services layer securely loads the operating system layer, and the operating system layer securely loads applications. The system microcode layer reads the encrypted system services layer across the secure boundary. It obtains the key necessary to decrypt the system services layer, from the secure data storage areas in which it keeps cryptographic keys. It decrypts the system services layer, which upon decryption is in plaintext within the secure boundary. It then computes a MAC (or MDC) from the plaintext and compares it with the expected MAC (or MDC) for the system services layer. If the MAC (or MDC) matches, then the system services layer is presumed to be trusted, and the system microcode layer passes control to the system services layer.

Obviously, this is a simplified description. At the very least key management would likely be somewhat more complicated. The basic premise is that the secure processor is able to securely store and use cryptographic keys that can be used to assure its own overall secrecy and integrity.

4.3.8 *Virtual Machine Monitor*

It may make architectural sense in some situations to implement strongly partitioned virtual machines rather than a single multitasking operating system layer. This might be done for relative simplicity of assurance or evaluation for example. With appropriate hardware support, virtual machines can be very strongly isolated from each other and from critical system resources. The system microcode layer is the highest reasonable layer to install robust virtual machine support. If a virtual machine is devoted to each application, and the virtual machine support is trusted to correctly isolate virtual machines, then it might be possible to relax security (and function) requirements for the operating system loaded into each virtual machine. Any further discussion of such a secure processor implementation is beyond the scope of this document.

4.4 System Services Layer

4.4.1 *Definition*

This layer contains the code that would typically be in an OS kernel, and contains the full cryptographic support. This layer does not provide an OS interface to applications. This is deferred to the next layer. Code that is both not exposed to applications, and useful to several different operating systems, is a candidate for implementation in this layer.

---

2 This decision would probably be a design decision rather than a dynamic decision. Because the state of the system services layer includes some variables, it may not be sensible to perform a cryptographic integrity check if the system services layer has changed state since it was initially loaded.

4.4.2 *Physical Placement*

This layer would typically be loaded into RAM upon each power up. If the processor design uses RAM that persists through a power cycle, then this layer could be loaded upon demand of an agent external to the secure processor. Note that memory that persists through a power cycle should nonetheless be erased upon detection of tampering.

4.4.3 *Full Cryptographic Support*

This layer is where full cryptographic support is introduced. Comprehensive cryptographic support is useful for implementing much of the other function in this layer. The IBM Common Cryptographic Architecture [IBM90a] typifies the level of cryptographic support envisioned for this layer. We want to be able to support a wide range of commercial applications of cryptography, and to support the associated key management.

The availability of cryptographic support enables full cryptographic guarantees of the integrity and secrecy of any layers loaded on top of this one. (See "Introduction" on page 19) In particular, this layer controls the cryptographic guarantees of integrity and secrecy of the OS layer. It needs to use and perhaps control keys for that purpose. We suggest that the cryptographic support include key management that is useful and sufficient for most applications.

The cryptographic support needs to have access to the persistent memory within the secure boundary that is used to store cryptographic keys and related information. Access to this memory other than by the cryptographic support should be strictly controlled. In some implementations, it may be a design constraint that the key storage areas can only be accessed by the cryptographic support.

The cryptographic support may include key management services which directly assist in the management of the keys used to assure secrecy and integrity of software loaded into the secure processor. Such assurance is a higher-level cryptographic operation than is commonly provided by cryptographic support.

4.4.4 *OS Primitives and OS Independent Security Kernel*

This layer contains the primitives that the OS layer uses to implement the OS interface provided to applications. Optimistically, this layer would contain an intersection of kernel primitives that are useful in implementing several operating systems.

The LOCK project [SAYD89] proposes a security kernel that is largely independent of any particular operating system. It may be possible to do the same in the system services layer of a secure processor.

Some secure processor configurations will support multiprogramming, and in particular will support multiple concurrently loaded applications. In this event, strong controls on interprocess communication are of particular importance. Much of the potential strength of this architecture lies in its ability to isolate certain selected processes. This strength might be largely dissipated if interprocess communications were not carefully controlled.

4.4.5 *Communications with this Layer*

This layer may communicate with the system microcode layer, the operating system layer, and outside the secure boundary. Communications with the system microcode layer are through interfaces that the system microcode layer provides. Communications with the operating system layer are through interfaces that this layer provides.

This layer will need enough communications support to load the operating system layer. Security considerations may dictate that direct I/O be limited to this layer and the layers below this layer. Portability considerations may dictate that direct I/O be limited to the system microcode layer, the layer below this one. If a particular system services layer writes directly to the I/O devices, it may not function correctly with some future version of the secure processor hardware. In any event, this layer should supply communications support that is useful to the operating system layer.

4.5 Operating System Layer

4.5.1 Definition

The operating system layer exposes an operating-system-like interface to applications written for the secure processor. This layer is designed to simulate a particular operating system's system calls. This layer is intended to support straightforward ports of code written for a familiar OS. To application developers, it should appear to be a close replica of the host operating system, or of a familiar host operating system.

The operating system layer includes whatever code is not included in the system services layer but is nonetheless needed to support the interface exposed to applications. The boundary between the system services layer and the operating systems layer is ill-defined and may differ between implementations. In some implementations, it may be plausible to dispense with the system services layer entirely, instead implementing the cryptographic support within the operating system, for example as a device driver.

4.5.2 Physical Placement

The operating system layer would typically be loaded into RAM upon each power up. If the processor design uses RAM that persists through a power cycle, then this layer could be loaded upon demand of an agent external to the secure processor.

4.5.3 Tasking

To a large degree, the question of whether or not to support multiprogramming in the secure processor is constrained by compatibility with the simulated operating system. If the ability to load multiple unrelated applications into the secure processor is excluded, a security analysis becomes much easier. If only one program can be loaded at a time, and programs are always associated with a single subject, then only one subject can be active in the secure processor. If there is only one active subject, access control decisions become simpler.

Unfortunately for simplicity of analysis, modern operating systems are multiprogramming (if not multitasking). Any operating system simulated by this layer is likely to allow several programs to be loaded concurrently. As noted in "System Services Layer" on page 25, much of the potential strength of this architecture lies in its ability to isolate certain selected processes from the rest of the computing environment, and this strength might be largely dissipated if interprocess communications were not carefully controlled.

Processes in the secure processor will need to communicate outside the secure boundary. It may be sensible in many implementations to allow external agents to access applications loaded in the secure processor via a remote procedure call mechanism. If so, then the remote procedure call mechanism will have to be supported. In general, interprocess communications across the secure boundary will have to be carefully controlled, both for performance and security reasons.

4.5.4 Assured Loading of Applications

One function of the operating system layer is to facilitate the loading of applications into the secure processor. The operating system layer makes use of the cryptographic services provided by the system services layer to assure the secrecy and integrity of loaded applications before giving them control. (See "Introduction" on page 19)

The set of valid applications is likely to vary more than the other layers of this software architecture. There will need to be a rich set of administrative functions that give appropriate agents (perhaps individuals) the ability to specify what applications may be loaded, and under what circumstances. There will need to be mechanisms in the secure processor, perhaps in the operating system layer, to enforce these rules. Ultimately, whether or not a secure processor is able to load an application depends on whether it possesses the key needed to decrypt the application, but access control decisions need not be based on possession of decryption keys.

4.5.5 *Communications*

This layer provides an interface to applications, as described in "Definition". It also needs to communicate outside the secure boundary. To a large degree, the form and content of communications across the secure boundary will be dependent on the particular operating system layer loaded. As is noted in "Tasking", interprocess communications across the secure boundary should be carefully controlled.

4.6 Application Layer

4.6.1 *Physical Placement*

Applications are typically loaded into RAM upon the demand of any agent allowed to request the load of an application. Applications could persist through a secure processor power cycle if they were loaded into persistent memory.

4.6.2 *Definition*

The application layer is where code gets loaded that allows the secure processor to serve in one or more roles. Applications may be ordinary applications that have been migrated into the secure processor for security reasons, or they may be specialized programs that enable the secure processor to assume some role in the overall system architecture. For example, an application could be loaded into a secure processor that enabled the secure processor to become an authentication server.

The application layer will be very dependent on the structure and capabilities of the operating system layer. Whether or not the operating system layer supports multiprogramming or multitasking will have a large impact on the structure and nature of applications. Similarly, the degree of I/O support provided by the operating system layer will also affect the nature of applications. For example, if the secure processor has no direct connection with the user I/O devices such as the display and keyboard, support in the operating system layer for such user I/O might be excluded.

4.6.3 *What Applications can be Loaded?*

Typically, the decision of whether or not to allow an application to be loaded will be an access control decision, made in the secure processor or in some similarly secure environment. The secure processor will need to either hold or have access to the cryptographic keys used to assure the secrecy and integrity of applications. (See "Introduction" on page 19) As noted in "Assured Loading of Applications" on page 27, the access control decision is not necessarily a function of the possession of these keys.

5.0 Secure Processor Hardware Architecture

5.1 Introduction

This chapter discusses hardware architecture considerations for secure processors. Secure processors are computing systems in which a processing engine, main storage, read-only storage, cryptographic function, and an interface to the "outside", are placed inside of a physical security boundary[WEIN87]. A current example of a secure processor is the Transaction Security System.

The physical security boundary is used to prevent physical probing of, or tampering with the computing system. With the appropriate physical and logical interface, a secure processor can be used in such a way that secrecy and integrity of programs and data can be maintained in an otherwise unsecured computing environment.

It will be shown that in many ways secure processors can be viewed in the same way as any other computing system. There are however some significant differences. The largest differences between secured and unsecured systems lies in the areas of cryptography and access control.

To maintain security, many of the programs and much of the data that will be used by the secure processor must be stored in an encrypted state when outside of the physical security boundary. This requirement necessitates decrypting during input and encrypting during output. As will be discussed in the model section, the placement and performance of the cryptographic function is critical to overall performance.

Access control requires that programs access only the correct data and system resources. Measures must be taken to prevent inadvertent or intentional access to disallowed system objects.

A model of the secure processor will be presented, and each of the parts will be discussed. The model parts are: the processor, cache and primary storage, ROS and persistent storage, secure data storage, bus interfaces, and cryptographic services. After the model parts have been described, the communication pathways between the parts will be defined. Once the complete system has been described, each of the parts will be re-examined from a performance standpoint.

This document presents general architecture. Model and performance parameters will be presented without specificity to a particular implementation. These considerations can then be used by implementers to develop secure processors for specific system designs.

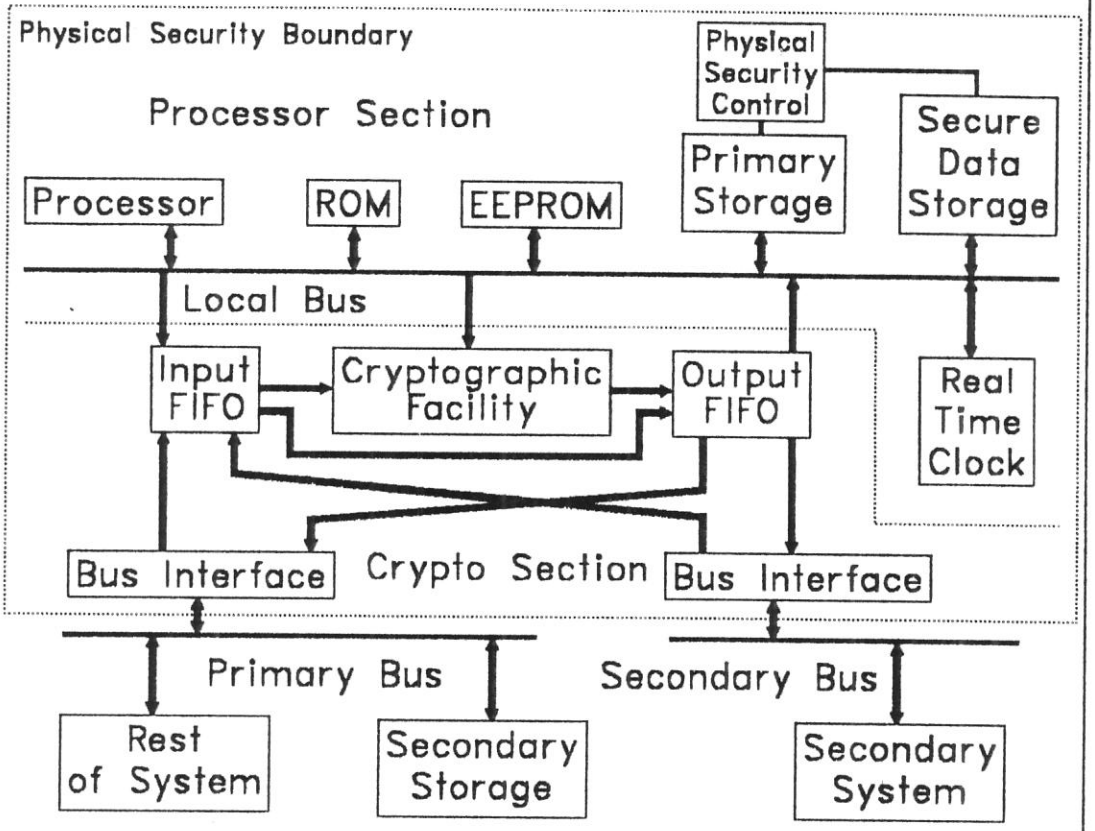

Figure 5. Secure Processor Architectural Model

5.2 Model

The model being described here is intended to show a complete array of functions required in secure processors. In many implementations, the model can be subsetted to meet cost/performance goals. The required functions are: the processor, primary storage, ROS and persistent storage, secure data storage, cryptographic services, and a primary interface. These functions can be implemented at whatever level is appropriate for the application. An 8051 smart card implementation is possible as is a 3090 implementation.

It is, however, assumed that the secure processor will be designed with a particular set of roles in mind as well as a particular target system. Options can be chosen or not to suit the particular needs.

Figure 5 shows the basic secure processor model. Variations of this model including subsets for smaller systems and options to increase performance will be presented.

5.2.1 Processor

The processor can be any processor capable of basic instruction execution and I/O. If the application requires that the secure processor be the main or service processor in a high performance computing system, the processor should have sufficient performance not to become a bottleneck in the system. Also, in general applications it is reasonable for the secure processor to execute the same native instruction set as other insecure processors common to the computing environment, but this is not required. The choice of processor instruction set is a development

decision. Of course, if the secure processor is replacing the main processor in a system, it should execute the same instruction set as the original processor.

The processor could be single or multitasking. If the system is a single tasking system which virtualizes storage or a multitasking system, paging or swapping must be cryptographically protected. Paging or swapping is moving part or all of a task's storage contents out to DASD and moving other information in from DASD. This is required when the processor switches from one task to another and needs storage space for the new task, or needs additional space for a particular task. Cryptographic paging is the same in that storage contents are removed to make room for other tasks. The difference is that the removed contents must be encrypted when exiting the secure processor and decrypted when retrieved.

The processor's most important requirement is that it control access to and use of cryptographic services, storage, and bus interfaces. These controls are required to prevent the loss of integrity or disclosure of secured data. This means that the processor must be able to limit access to I/O and storage locations to prevent programs from performing disallowed actions or accessing disallowed data.

5.2.2 Primary Storage and Cache

As was previously mentioned, data and programs used in a secure processor may need to be encrypted while outside of the secured boundary. Cryptographic services are a potential performance bottleneck. To avoid performance problems, primary storage has to be large enough to prevent severe performance degradation resulting from excessive paging. The optimal size can only be determined once the target processor, operating system, and expected function are known.

Caching of the primary storage can be done in any usual way providing standard assurance precautions are taken to prevent access to residual data. This is to say that data left in the cache by an exiting task should not be available to another task.

A performance enhancement option is dual porting the main storage. This would allow cryptographic services to page data in or out while processing continues. This would reduce performance problems caused by slow cryptographic hardware.

5.2.3 ROS

The function of the code stored in ROS is to bootstrap and to perform an IML of the processor. ROS can also be used to hold interrupt vector tables etc.,to guarantee their integrity. A low-level self-test to assure function at the lowest level should also be included.

5.2.4 Persistent Storage

The persistent storage is used to contain the microcode layer, which implements the lowest level cryptographic and interface code. This code could be loaded by the manufacturer or by the system administrator. The main purpose of this code is to permit cryptographically secure loading of the system services layer.

During IML a complete diagnostic routine should be performed. These tests should check all of the secure processor functions, including the access control, for full compliance to specification.

5.2.5 Secure Data Storage

In secure processors, some amount of persistent volatile storage is required to store cryptographic keys and other secret data. This storage is commonly battery-backed RAM, whose power is controlled by the physical security circuitry. In the event of physical tampering this data would be erased to prevent disclosure. Access to this storage must be strictly controlled and is only granted to security kernel functions, i.e. encryption and decryption services and key control. The amount of secure data storage required is usually very small compared to program storage, and could be paged if necessary.

5.2.6 *Interfaces*

The interfaces connect the secure processor with the outside. In addition, the interface must ensure that no "leakage" of internal signals occur. This means that when the interface is closed, no small signals or level variations which would indicate state or process can be detectable from outside the secure processor.

The form of the interface can be whatever is required as long as it meets the previous requirements. From Figure 5 on page 30, it can be seen that in a full implementation the interfaces are triple ported. This is done to allow pipelining of data from the primary bus to the auxiliary bus, with or without cryptographic conversion. This is valuable where the secure processor functions as a device controller and can improve performance. This secondary interface and this pipelining feature are optional and may not be required in all applications.

Implementing individual control processors for the interfaces would be necessary for the bus interfaces to move or pipeline data without intervention of the processor. This allows the highest performance with given hardware.

5.2.6.1 Primary Interface

This interface is the primary connection to the rest of the system or network. This interface can be tailored to any required communication protocol. If programs are to be loaded into the secure processor or the secure processor is to be used as a data path controller, the bandwidth of this interface becomes important. If all instructions are found in ROS, and the volume and frequency of accesses are small, the speed is less important.

Buffering the interface should be considered if speed is an important issue. This will allow transfer of data while allowing the cryptographic facility and processor to continue working.

5.2.6.2 Secondary Interface

This interface can be similar in function to the primary interface, or it may be used for protocol conversion from one bus to another. In small implementations, it may be left out. Its main value is to optimize the secure processor for use as a channel or I/O bus controller. It has the same security restrictions as the primary interface in that when closed it must prevent all data leakage, and access to it must be controlled.

5.2.7 *Cryptographic Services*

Cryptographic services will be provided for protection of data. Where the secure processor is to be used in high performance systems this is the greatest potential bottleneck. Buffering the cryptographic services will help to relieve overly high bus utilization and get the highest performance.

5.2.8 *Real Time Clock*

There are many functions, including audit and access control that need accurate time and date information. A real time clock, backed-up by the battery which backs the secure data storage should be included within the secure processor boundary. Write access, to set the clock, should be considered as important as access to the secure data storage and access should be controlled accordingly. Read access could be at the system service, operating system, or application level.

5.2.9 *System Communications and Control*

All of the elements of the secure processor are connected to a local bus which is located totally within the secure enclosure. The processor and other devices connect to the bus in a conventional manner. The control structure must support positive access control. This access control must extend to storage as well as other hardware devices. Tasks or programs running on the secure processor must be able to be isolated from each other. Only the security kernel can grant or change access. In high performance processors, access control is frequently part of the processor's design and can accomplish this function directly. In lower performance processors, access control to

storage areas or I/O will have to be handled by additional hardware. Whatever method is chosen, the control must be positive and fail-safe.

5.3 Specific Hardware Requirements

While the secure processor model is quite general, there are some specific requirements that must be fulfilled. These requirements fall into three areas: storage access control, I/O access control, and interface isolation.

Storage and I/O access control are "built in" to many high performance processors. Such processors are fully able to meet the needs described here. Implementations using lower performance processors will need to add these functions in external hardware or be limited to only loading and executing trusted functions.

5.3.1 Storage Access Control

Each of the programs which will execute on the secure processor needs to access a number of different kinds of storage. Primarily they will need to access the storage in which they reside, in order to execute. Programs may also need to access high or low level system services in persistent or main storage and they may need to access their own secret data in the secure data storage. However, all programs can not be able to do so. Arbitrary applications which will be run in the secure processor may not necessarily be programs which are trusted. To maintain security, the security kernel must be able to control what storage a program can access. In general, a program should only be able to access its own program and data areas. It specifically should not be able to access the security kernel area, the secure data storage, or another program's area without specific permission from the security kernel.

5.3.2 I/O Access Control

I/O access control refers to accessing the interface(s), the cryptographic services, the real time clock (if present), or other I/O devices on the secure processor's local bus. As above, access to any of the I/O devices must be controlled by the security kernel.

5.3.3 Interface Isolation

To maintain the secrecy of the processes in the secure processor, it must be impossible to read any information from the secure processor that is not intentionally sent out. The most likely place that information would leak out of the secure processor is at the interface(s). Not only must the interface be prevented from being open when it should not be (an access control function), but small signal leakage must also be prevented. Small signal leakage refers to small variations which occur on outputs due to changes on inputs, when the outputs are in a blocked state. An example of this is to change the input state on a D type flip-flop without clocking. Small variations of the output may occur which will tell the state of the input. Conditions like this can allow information to be read through closed latches. This would, of course, violate security.

In implementing a secure processor care must be taken to prevent information leakage from occurring. Multiple level buffering of data or opto-isolation may be required.

5.4 Performance Considerations

In some cases, the performance of the secure processor will not be an important issue. In many cases it will be important. The throughput of a system may depend on the speed at which a secure processor can provide a service or cipher data. At each point the implementer will have to consider the cost/performance trade-off. Storage cache for the processor can dramatically improve performance, but it is expensive. Cryptographic services can be accomplished in software instead of hardware, at a much slower rate; but the cost reduction may be substantial.

5.4.1 Processor

It is obvious that in any computing system the processor sets the limit for performance. The processor chosen for a particular implementation needs to address the requirements of the intended application(s). If the secure processor is to be the main processor, it should have at least the same performance of the processor that it replaces. If the system has a bus speed higher than the cryptographic rate, it should probably have slightly higher performance to offset delays caused by en/decryption. If the secure processor is intended as a service machine, where secure functions or arbitrary programs are loaded and executed, having the secure processor use the same native instruction set as the client is useful. However if the secure processor is a fixed function service machine, such as a Smart Card, where the instructions remain constant, the native instruction set is not as important, and becomes a development issue. There are valid reasons for the secure processor's native instruction set to be, or not be, the same native instruction set as the system the secure processor will operate with. For that reason there are no architectural requirements for the native instruction set of the secure processor. This issue can be most effectively determined in development and is a design decision.

5.4.2 Cryptographic Services

As has been stated previously, cryptographic services are the largest potential performance obstacle to be encountered. Cryptographic services can be implemented as pictured in Figure 5 on page 30, or they can be implemented in software. If the software route is chosen for cost reasons, performance impacts in addition to the rate differences need to be understood. First the data to be ciphered needs to be loaded into processor storage. This takes some time even before cryptography can begin. Then the data can be ciphered. While it is occurring, the cryptography is consuming a large amount of computing resources that would be available to other tasks. Once this is accomplished the data can be used.

If hardware encryption is used, the time is reduced. The time required for encryption is the time needed to initialize the cryptographic device, plus the time required to load then cipher the data. Cipher/load can be done in one operation, or the data can first be loaded into a buffer and then ciphered. In higher performance systems where the bus is faster than the cryptographic device, the cryptographic process will use a greater than necessary fraction of the bus resources to transfer the data. This is why buffering is useful: the data can be moved quickly, faster than the cryptographic device can accept it, then the cryptographic service can continue without wasting system resources.

The efficiency of data transfer, as a function of transfer rate, can be viewed in the following way:

$$Xfer\ efficiency = \left(\frac{Rate_{xfer}}{Rate_{Bus\ max}} \right)$$

Where:

$Rate_{xfer}$ is the data transfer rate.
$Rate_{Bus\ max}$ is the maximum bus transfer rate.

It is of course most desirable for the transfer rate to be the maximum at all times. That way it will take the minimum bus resource to move a given amount of data. Buffering will not increase the cryptographic rate, but it will decrease bus utilization and will allow the processor(s) (the secure processor and the sender/receiver) to spend less time in data transfer. If the secure processor interface is buffered, the host processor can transfer data to the secure processor at the highest speed for maximum efficiency, and the cryptographic device can then work at its own (slower) pace. This is also applicable in the reverse direction. If the same preservation of the secure processor's time is desired, then a buffer can be placed between the secure processor's local bus and the cryptographic device. The secure processor can then move data from the cryptographic buffer to storage, or the reverse, at its maximum rate, and with minimum resource utilization. Figure 6 on page 36 shows the processor model with the addition of the buffers. The transfer time for a data block is developed in the following equations.

The total number of packets in a data block is the total number of bytes divided by the number of bytes in a single packet.

$$\frac{Datasize}{Bytes_{packet}} = N_{packets}$$

Where:

$Datasize$ is the total amount of data to be moved.
$Bytes_{packet}$ is the amount of data per single transfer,
$N_{packets}$ is the total number of packets.

The time required to move a packet is the packet size divided by the byte transfer rate.

$$\frac{Bytes_{packet}}{rate_{xfer}} = T_{packet}$$

Where:

$Bytes_{packet}$ is the amount of data per single transfer,
$Rate_{xfer}$ is the transfer rate.
T_{packet} is the transfer time for a single packet.

The time required to en/decrypt a packet is the packet size divided by the cryptographic rate.

$$\frac{Bytes_{packet}}{rate_{crypto}} = T_{crypto}$$

Where:

$Bytes_{packet}$ is the amount of data per single transfer,
$Rate_{crypto}$ is the cryptographic rate.
T_{crypto} is the crypto time for a single packet.

The time per packet will be the time it takes for the cryptographic device to empty/fill the buffer, plus the waiting time until the next fill/empty begins, plus the time it takes to fill/empty the buffer from the bus, plus bus overhead. This time multiplied by the number of packets plus the crypto setup time gives the total transfer time. F_{sync} represents the overlap in transfers permitted by a dual-ported asynchronous buffer. If the buffer is fully asynchronous, the slower function will totally dominate the time. I.e. if the bus is faster than the crypto unit and the buffer is asynchronous, the cryptographic time will dominate, and the bus time will be transparent.

$$N_{packets}(F_{sync}(T_{packet} + T_{overhead}) + T_{crypto} + T_{wait}) + T_{setup} = T_{transfer}$$

Where:

$T_{overhead}$ is the setup and release time per packet.
T_{wait} is the waiting time between the buffer emptying and the next transfer beginning.
T_{setup} is the setup time of the cryptographic device.
$T_{transfer}$ is the transfer time
F_{sync} is a synchronicity factor for the buffer where $0 \leq F_{sync} \leq 1$. A value of 1 corresponds to a fully synchronous buffer where only one port may be active at one time. A value of 0 corresponds to a fully asynchronous buffer where both ports may be fully active at all times. Other buffer designs will need to have F_{sync} determined heuristically. This assumes that the cryptographic rate is slower than the bus rate. If the reverse is true, F_{sync} is multiplied by the T_{crypto} term.

Minimizing the number of packets will minimize the overhead caused by starting and stopping the transfer. A special case which could arise in certain applications, is the case of many small transfers. In that case the cryptographic setup time can become dominant and the cryptographic rate less important.

The size of the buffer will be system dependent. In multitasking systems the size of one storage page would be useful. A page is the standard amount of storage that is switched when space is needed. A page varies in size from 128 bytes to 4K bytes typically, with 2K or 4K bytes being very common.

In some systems access to the bus is limited to some finite fraction of the bus resource, or to some amount of contiguous time. This is done to make sure all of the system devices are able to "get their chance". In systems with limited time or fractional access to the bus, a buffer which would hold one transfer's allotment of data would be a good choice. Ultimately the buffer size is a trade off between cost and bus utilization, with $N_{packets}T_{overhead}$ and cost being the major variables. The larger the number of packets that a block must be split into to complete the transfer, the greater the amount of bus total utilization.

The bus usage, for the duration of a block transfer, will be the fraction:

$$\frac{N_{packets}(T_{packet} + T_{overhead})}{N_{packets}(T_{crypto} + T_{wait}) + T_{setup}} = Fractional\ Bus\ Use$$

This equation shows what fraction of the bus resources are used to perform a cryptographic operation. The previous equation shows how long this usage lasts. Together these equations can be used to determine what portion of the system resources are needed for cryptographic operations. That information can be useful in determining overall system performance. The smaller the T_{packet} the smaller the fraction of the bus required. The smaller the T_{crypto} the smaller the latency to the availability of data. T_{wait} should be minimized, as this is just wasted time.

System parameters and cost will be the most important considerations in determining if a buffer should be used, and how big it should be.

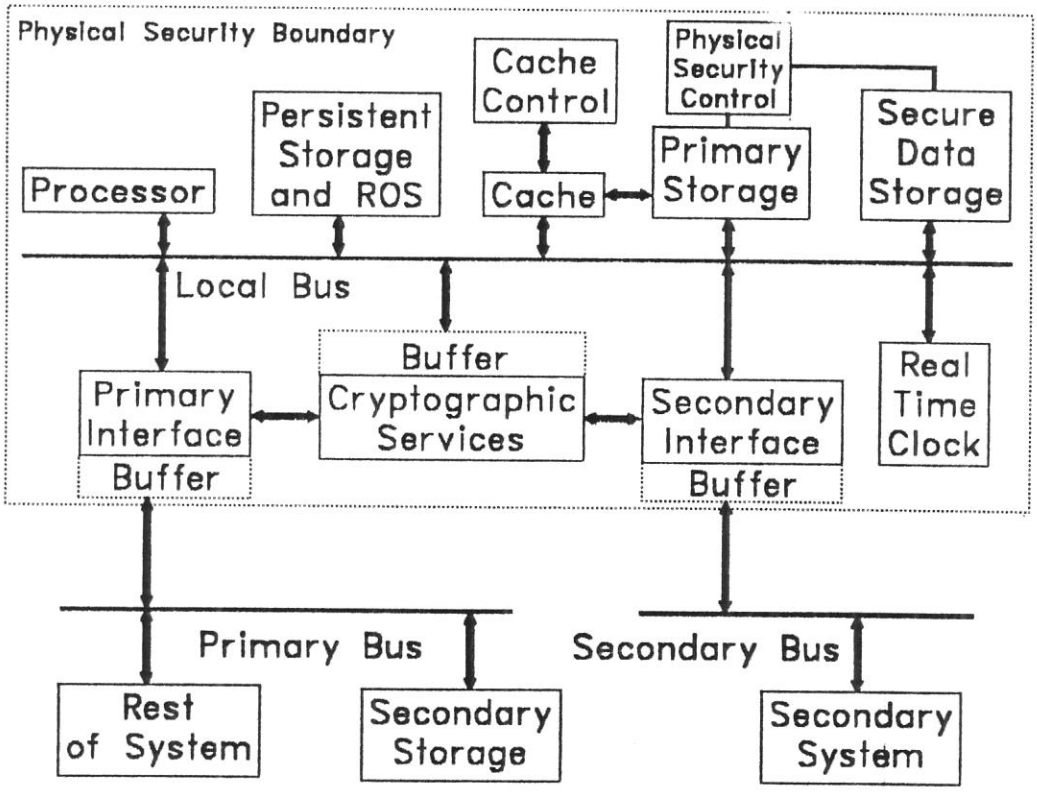

Figure 6. Secure Processor Architectural Model with Buffering

5.4.3 Storage

In general, storage considerations are similar to the processor considerations, in that the use and application will largely determine the type and amount of storage. Each of the storage areas has to be considered for size and performance.

5.4.3.1 ROS

The ROS will not need to be very large, and the code contained in it will only be used during power-up and IML. Outside of reliability, its performance is not critical.

5.4.3.2 Persistent Storage

Since the persistent storage contains the low level interface code for the secure processor, it will be used often in execution and should not be slow storage. It will contain test, system service, and interface code, so it will need to be of a moderate (BIOS) size. Persistent storage needs to be re-writeable but reliably persistent. EEPROM or Flash EPROM are probably the best choices for systems with sealed physical security enclosures.

ROM and EPROM could also be used in systems with openable enclosures. If this approach was used updates would be applied by changing the physical devices.

5.4.3.3 Primary Storage

Primary storage will contain the system services, the operating system, and applications. It can be any type of RAM otherwise suitable for the purpose.

5.4.3.4 Secure Data Storage

The secure data storage must be maintained at all times regardless of the power state of the rest of the secure processor. Since it contains the system en/decryption keys and other secret data, the maintenance and power back-up systems must be reliable.

5.4.4 Storage Requirements with Active Physical Security

Physical security is discussed in "Physical Security" on page 42. If active physical security is used, there are additional requirements placed upon all elements in the secure processor that store any secret information. The primary storage, secure data storage, and any buffers must be able to be erased in the event of a break-in attempt. In addition all other devices, such as the cryptographic device, which may contain secret data must also be erased. This may include the main processor. The erasure time should be in the 1's to 10's of milliseconds, depending on the rest of the physical security design.

5.5 Secure Processor Placement

One of the most relevant questions in implementing a secure processor is: Where should the secure processor reside in the system? The heavy boxes in Figure 7 on page 38 show a number of examples of where the secure processor could be placed. Each of these locations has unique advantages and disadvantages concerning utility and system performance.

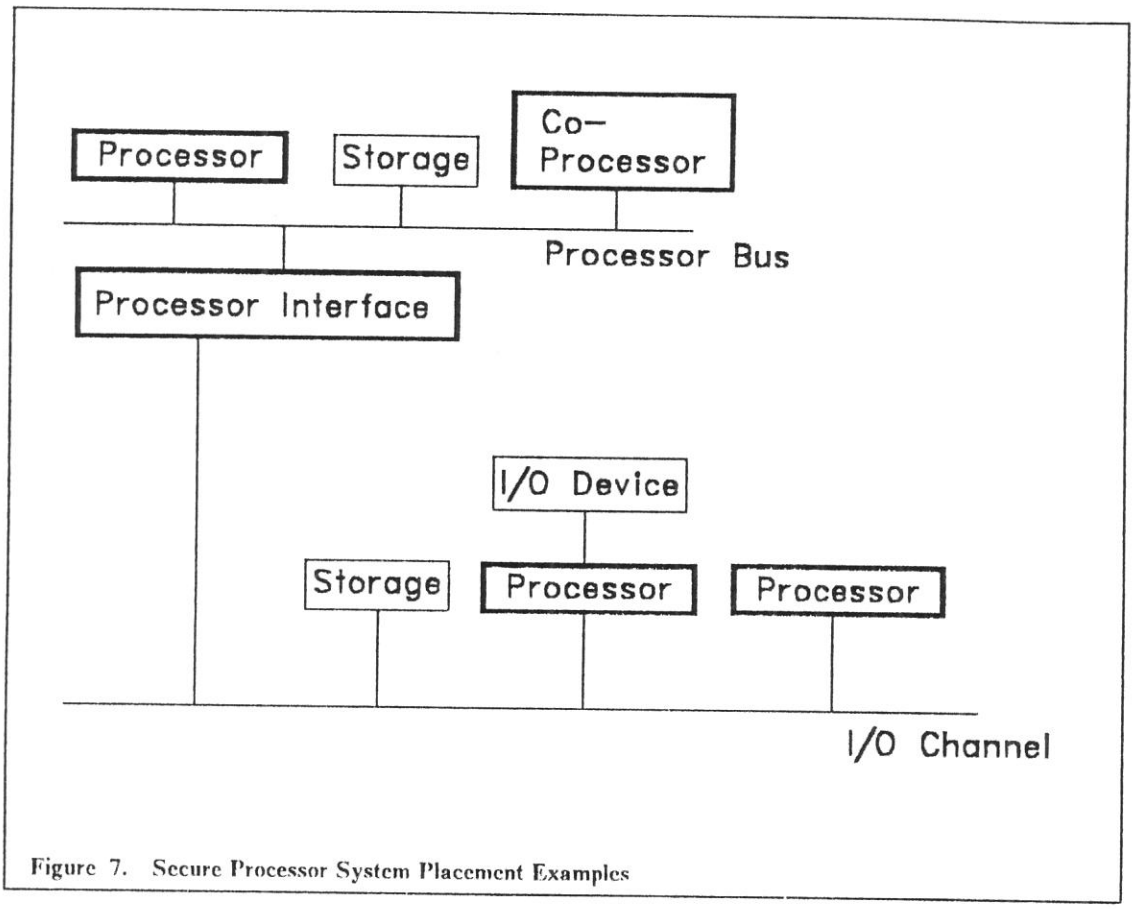

Figure 7. Secure Processor System Placement Examples

5.5.1 Placement as the Main Processor

If the secure processor is the main processor in a computing system, programs can be run in the protected storage within the physical security boundary. If their security is not critical, they can be run in external storage on the primary or secondary bus.

If the programs are to be run in internal storage, on the secure local bus, that storage must be of sufficient size to prevent performance losses due to the limitations of cryptographic paging. Too little storage may not only cause extra paging, it may also cause the use of an excessive portion of the bus capacity. This can happen when large, slow transfers cause excessive overhead due to repeated starting and stopping of transfer, or by using the bus at less than full speed for long periods of time. This can affect overall bus availability and can slow the execution of a specific program due to paging delay. Buffering the interface and the cryptographic facility can help to alleviate this problem. This can be accomplished by allowing all bus transfers to occur at full speed, followed by cryptographic services occurring effectively off-line at a slower rate. The problem of limited bandwidth cryptographic devices (typically $\leq$1.7 Mbyte/sec. for currently available commercial parts) will affect all but the lowest performance systems if pages or bulk data need to be encrypted. This problem will remain until faster devices are produced. If integrity is a concern as well as secrecy, there is another requirement in addition to encryption. Programs and data may also have to be cryptographically checksummed to ensure that the data has not been altered. Most methods of doing cryptographic checksums securely require multiple cryptographic passes and are therefore very costly from a system resource standpoint.

If programs are to be run in external storage, the interface must be arranged to ***open*** and ***close*** at each appropriate instance. These instances occur, for example, when a program executing in primary or secondary storage makes a call to the security kernel. When the processor shifts state to the security kernel, the internal bus must be cut off from the primary or secondary to prevent

security kernel code and data from appearing on the external bus. During these periods, support for memory refresh, and any active external processes (i.e. DMA between peripherals), must be maintained externally to the secure processor.

If the secure processor is the main processor, the greatest advantage is that all code executed on the system can execute in a protected environment. Audit and access control can have greater, possibly absolute, assurance for all system objects as well.

The largest disadvantages are all performance related. Until faster cryptographic devices become available, a system that makes extensive use of cryptographic protection for files and paging will have serious data transfer rate limitations. Extensive use of auditing facilities will also impact performance. In systems which use current secure operating systems, auditing and access control functions can take as much as 20% of the processor's time.

5.5.2 *Placement as a Co-Processor or Service Processor*

If the secure processor is to be used as a co-processor or service processor in a computing system, the situation becomes somewhat simpler. The secure processor can be thought of as an intelligent peripheral residing where appropriate within the larger system. During IPL, the secure processor's kernel is provided by the main system, after which the secure processor can operate on data and execute code securely as a service to the system. If the secure processor needs additional code or system information, it requests the service. Performance considerations hinge totally on what kinds of requests are to be processed, their size, frequency, and acceptable latency.

In this placement, the secure processor can reside on the processor bus or on a local or remote I/O channel. If the secure processor resides locally, it has the fastest access to the processor and its associated storage. This could be advantageous for audit, access control, or authentication services. It has a longer path to I/O and mass storage which could be a disadvantage. Another potential performance problem associated with this placement is possible interference with the main processor while accessing the system resources.

If the secure processor resides on an I/O channel, it benefits from direct access to system resources. It loses direct access to processor resources, such as local storage, and the communication path to the system processor becomes longer. Access control and audit performance will probably improve as there will be less competition for local resources, especially the main processor's local bus. The biggest disadvantage of this placement is that the secure processor can not always see what the main processor is doing. Because of this, the secure processor will not be able to audit or monitor the main processor's actions directly.

This placement model can also be viewed with the secure processor as a remote server device attached to an I/O channel via a communication link. This is probably the best know server model and is accurate for scenarios such as name or authentication servers[STEI88], or smart card authentication.

5.5.3 *Placement as an I/O Device Controller*

In this placement the secure processor maintains the characteristics of a server on an I/O channel and gains the advantages of being a gateway to an I/O device. The secure processor can serve as a hardware access controller giving greater assurance to access control and audit of controlled device. Good candidates for the I/O device are communications and DASD. Pipelined cryptographic services are possible. At current data rates, DASD or LAN access could proceed at full or nearly full speed with cryptographic protection, at least on workstations. Access control to a DASD unit could prevent access to or erasure of file objects, even without a secure operating system.

5.5.4 *Placement as an I/O Bus Interface*

The use of a secure processor as an I/O bus interface allows access control and audit for all devices on that I/O bus just as it did for a single device. The secure processor can now control access to and audit all of the devices on that bus. Once again performance limitations are the largest potential

problem. If cryptography is not required for all of the data, the problem might not be as severe, and access control and audit are still available.

5.5.5 *Parallel vs. Serial Device Placement*

An additional consideration for secure processor placement/system performance is the additional bus usage that comes with parallel placement. In example *A* in Figure 8 data handled by the secure processor on its way to the system from a device, or the reverse, must be put on the bus twice. Once from the device to the secure processor, and a second time from the secure processor to system. This results in additional bus resource utilization and additional latency. In the worst case, the bus could become saturated, and the overall system performance will drop as devices get locked out of the bus.

In example *B*, the secure processor acts as a pipeline between the device and the rest of he system. Any data from the device that is sent to the system only has to be placed on the bus a single time. The latency may not be appreciably shorter than in *A*, because the data still has to come from the device through the secure processor and onto the bus. The data still has to travel on two busses, but only once on the system bus. This is a significant advantage of using the secure processor as a device controller.

The secure processor in example *B* can still service other system devices in the same way it could in example *A*. However the data will have to travel twice, just as it did in example *A* when used in that way. The flexibility of a fast path for a critical device and slower paths for small/less critical tasks is gained. In applications where the device exchanges a large amount of data with the system, this is a very useful resource savings. Devices like DASD or LAN adapters are good candidates for this configuration.

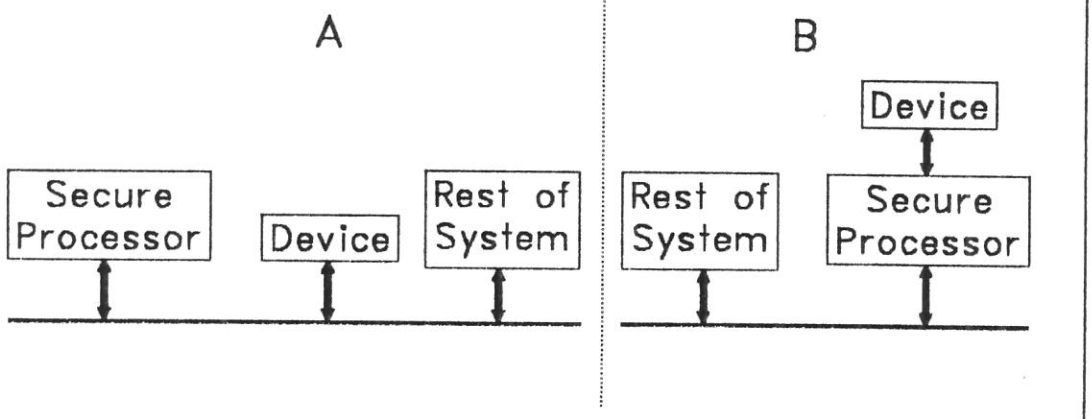

Figure 8. Parallel vs. Serial Placement Examples

5.6 Bus Monitoring

Bus monitoring is a common application of co-processing security devices. The following describes some of the techniques and uses of these devices.

Bus monitoring or using a "watchdog" processor are methods of securing a processing system by placing a hardware device in the system which watches all of the system operations. This observation can include each instruction execution. This method can prevent any disallowed

functions from occurring, or at least audit their occurrence. There are some serious limitations placed upon systems which adopt this method.

The monitor device must be able to control or audit the target system in real time. This requires the monitor to be significantly faster than the system in which it resides. That places an upper bound on the performance of the target system. It also places a cost requirement on the system for the monitor device which is potentially very high.

There are applications, specifically in the military, where there is a need for this type of system; but this may be excessive for the types of systems described here.

A type of watchdog monitor which can be very useful has been implemented in the IBM AS/400 computer. It is a monitor that watches for system exceptions or errors. If an error condition is found, the error and the system state are recorded to facilitate understanding and correction of the problem. This type of monitoring can be valuable for maintaining system integrity.

There are simpler types of monitoring which can be useful in the types of systems discussed here. Monitoring of a small number of critical system addresses outside of the secure processor can be useful in access control and audit of system resources. This might include monitoring the addresses of the DASD or LAN adapter and auditing access or causing a system exception to control access. This method has been used successfully in at least one commercial product, Tri-Span by Micronyx [NCSC87b].

At this time bus monitoring is not part of the Citadel architectural considerations. This is due to the potential high cost and complexity of implementing such a method on multi-bus high performance systems. These systems make up a significant number of the systems targeted by this architecture.

6.0 Physical Security

6.1 Introduction

Security for computing systems has two components: logical security and physical security. Logical security is concerned with the use of passwords, the security of the operating system, and so on. Physical security is concerned with the difficulty of tampering with data by means of hardware tools.

The purpose of both logical and physical security is to deter attacks on the valuable information assets of a computing system. There is no way to completely prevent all attacks on a system in every possible circumstance. Any security system can be penetrated, given sufficient time, effort, and skill. But this does not mean that security is unimportant. A bank vault that can be burgled with great effort, and only by a few highly skilled people is not useless. On the contrary, it does a good job of deterring attack, and of limiting the risk of attack to an acceptable level, which is precisely its purpose.

A good security system must deter a wide variety of attacks. Just as a house whose door is locked but whose windows are open is not secure against burglary, a computing system which is logically secure but physically vulnerable is open to attack. A well designed security system must achieve a balance of logical and physical security which, when combined with business policies, reduces the risk to the assets it contains to an acceptable level.

We propose that physical security systems be classified according to criteria that reasonably ensure the difficulty of mounting a successful attack against them. Difficulty is measured both in terms of the kinds of equipment, and the kinds of skills, required to complete the easiest known attack. Systems that require more specialized tools and skills are considered more secure since they limit the group of people who will attempt the attack. This degree of difficulty is further ensured by requiring good documentation, testing, quality assurance, and so on. Physical security systems which are designed to strongly deter attacks are required to meet higher levels of assurance in these other areas as well, so that design and implementation flaws do not jeopardize the protection given.

It is not possible to determine the suitability of a physical security system without understanding the value of the assets it is intended to protect, and the environment into which it is to be placed. A bank which intends to protect the contents of its safety deposit boxes does so by placing them inside of a massive vault, and by controlling who has access to the vault. Similarly, a company which wishes to protect extremely valuable data in a computing system should use a physical security system which offers substantial deterrence to an attacker, and should control who has access to the system in the first place.

6.2 An Overview of Physical Security

The term physical security has been used to discuss protection from a whole class of damages a computer can suffer. Lightning, water damage, and theft of the whole system are probably the most common topics considered. From these points of view, the system is the major asset and the data is usually not the prime concern. Here we consider the value of the information stored in the computer.

For the purpose of this document, physical security will be defined as a barrier placed around a computing system which prevents an unauthorized individual from physically accessing the computing system.

6.2.1 Kinds of Attacks

The range of physical attacks go from the simple theft of an unsecured tabletop system, to sophisticated attacks which decipher the radio frequency noise emanating from a system a block away. The tools used for constructing and testing computers can be used to read or modify

information within them. The increased quality and portability of both electronic and mechanical tools allows for a higher level of sophistication of attacks.

Ultimately the attacker has the goal of achieving some gain by attacking the system. This gain could be the revelation of secret data, which is called data theft. It could be modification of data, which is an integrity attack. The hardware could be stolen for its own value, which is called hardware theft.

6.2.1.1 Data Theft

Data theft can be subtle. Data can be copied without physically removing anything from the system. There are a number of straightforward ways to perform an attack of this type. If the system is logically secure so that the attacker cannot get the data via keyboard commands, the system can be electronically probed to gain the data.

For example, a system which is protected by encryption of data on the disk, would not suffer a data theft loss if a copy of the disk was taken. If an attacker was able to monitor the data bus in the system with an analyzer and find copies of the cryptographic keys, then the attacker could successfully steal the data.

An additional problem with data theft is that it can be accomplished without touching the system. There are instances where data can be captured from a distance of over a hundred feet and the act would remain undetectable. This is accomplished by receiving the radio frequency emanations from a computer, then reconstructing the original information. This kind of attack is known as TEMPEST. This is a difficult attack because complicated equipment and a high degree of skill are required to accomplish it. But if the value of the data or the motivation to attack it is sufficient, it can be done reliably.

To protect data from theft, physical access to sensitive information must be controlled at a level sufficient to deter or prevent theft. This can be accomplished by logical access control and a physical security system. The level of protection required is based on the value of the data, and the access an attacker might have. The object is to make the risk, cost, and difficulty not worth the effort required to obtain the asset.

6.2.1.2 Modification

Modification is more difficult than theft, but if it is accomplished successfully, information can be changed without the user/owner's knowledge. Examples of this type of attack include capturing the password with an analyzer, as above, then changing the contents of the protected data. Another method would use a logic injector or sequencer to modify storage or DASD contents directly.

A system in which the content of the data, and programs, can't be trusted is effectively useless. Again as in protection from theft, the data must be protected in a way that deters and resists tampering.

6.2.1.3 Hardware Theft

In the case of small systems, removal of the whole system is the easiest attack. In both small and large systems, parts of the system, such as DASD or other valuable components can be removed. These attacks can be combined with data theft. The contents of DASD etc. are removed along with the device itself. The results of a hardware theft attack can be far reaching. The hardware is lost, the software may be lost, and the user may now be faced with denial of service due to the unavailability of the system or data.

6.2.2 What Can Be Done to Protect From Attack

One of the most effective methods of preventing attack or tampering is control of the system environment. Preventing access to the system by the attacker is the most important step. If a system can be kept in a secure area, where individuals need to be identified and logged to gain access, the system is somewhat protected from the start. A system should be placed in the most protected environment feasible.

Even when a protected environment is available, it is not always sufficient protection. Even in the most secure environment, greed and blackmail can't be discounted. A highly secure environment will provide a level of protection by limiting access to some degree, but as the value of the asset increases, so do the motivations to deny, steal, or modify them.

In other cases, items of value must be placed in insecure or public environments. ATMs (Automatic Teller Machines) are probably the best known example of systems with valuable assets which are placed in a public environment. The money contained in the ATM is only part of the system's value. Customer identification data and cash disbursement and collection data can easily exceed the value of the money in the machine. ATMs are logically secure in that encryption and other security techniques are used in communication with other systems. This prevents forgery or modification of data logically. ATMs are also physically secured. The most notable physical security feature of an ATM is the vault-like construction (known as tamper resistant, explained below). Additional physical security features are usually less noticeable. Alarms may be installed in the ATM to detect tampering or excessive vibration which would sound if drilling or hammering was detected. Dye sprays can be placed in the cash box to mark the money in case of a detected theft attempt. And a mechanism could be installed to erase customer data to prevent its theft or modification in the case of an attack upon the electronics.

There are useful methods available to protect computing systems from physical attack. The important part in choosing such a method, is to choose a system appropriate for the environment and the contained value.

6.2.3 Kinds of Physical Security

A number of physical security methods are currently in use or in development. This is a new field in the commercial realm and is still being developed. The U.S. government has been working on this problem for almost 25 years, but the results remain classified. The ways and means described here, are not by any means an exhaustive list, nor are they represented as ultimate methods. Development is continuing in protection methods, and it is proceeding as well in attack methods. Any evaluation of appropriateness of a physical security system is time dependant, and must be repeated at intervals.

6.2.3.1 Tamper Resistant

Tamper resistant systems take the bank vault approach. This type of system is typified by the outer case design of an ATM. Thick steel or other tough materials are utilized to slow down the attack by requiring tools and great effort to breach the system. This type of system can be used in many environments and sometimes has the advantage of being so physically heavy (as in ATMs), that it resists theft by sheer weight. A system that is only tamper resistant has the disadvantage that the owner may not be aware of the loss until the break-in is discovered. That may be never, if the attacker did a "neat" job and replaced any material that had been removed with an exact duplicate.

6.2.3.2 Tamper Responding

Tamper responding systems use the burglar alarm approach. Defense is detection of the intrusion, followed by response to protect the asset. The response may consist of sounding an alarm, in the case of attended systems; and erasure or destruction of secret data to prevent its theft, in the case of isolated systems, which cannot depend on outside response. Tamper responding systems do not depend on robust construction or weight to guard an asset, and are therefore good for portable systems, or other systems where size and bulk are a disadvantage.

Another advantage of tamper responding systems is that they may not require attendance in the case of an attack. The response can be predetermined in the design (such as erasing the secret data). In the event of a detection, the response is engaged and carried out. Disadvantages of tamper responding systems are false alarms and misses. In the event of a false intrusion detection, valuable data could be erased needlessly. In the event of a miss, valuable data could be stolen.

6.2.3.3 Tamper Evident

Tamper evident system are designed to make sure that evidence of a break in is left behind, if one occurs. This is usually accomplished by chemical or chemical/mechanical means, such as a white paint that "bleeds" red if cut or scratched, or tape or seals that show evidence of removal. This approach can be very sensitive to even the smallest of punctures. These systems are not designed to prevent an attack, or to respond to the indication that one is in progress. Their job is to ensure that the fact of the break in will remain known, and can be ascertained at a later time.

The advantage of tamper evident systems is that they can be inexpensive. The disadvantage is that these systems do nothing to prevent attack, they record the event and depend on effective physical auditing procedures to recognize attacks. The type of data that is allowed in these systems is also important. Data with long term value that could not be invalidated, would in most cases be inappropriate. Data with dated value, which could be de-valued before it was used, could be acceptable.

6.3 Choosing or Designing a Physical Security System

The discussion of physical security here is a very basic outline of the need for physical security and the way it may be implemented. Very careful evaluation of the asset value, environment, and physical security must be made to determine the exact needs of any system. For greater detail refer to the companion document,"A Physical Security Evaluation System" [WEIN88]. For an example of the current state-of-the-art commercial physical security system, examine the Transaction Security System.

7.0 Implementation Examples

In this chapter, we give examples of how secure processors could be implemented in a variety of systems. Each implementation conforms to the architecture introduced in this document. Implementations on increasingly smaller systems may use subsets of the full architecture, since not all of the functions of the full architecture are necessary for these applications.

7.1 Within Single Systems

First, we consider how a secure processor can be incorporated into three different systems: workstations, mainframes, and smart cards. We begin with workstations because it is the example most familiar to us.

7.1.1 Workstations

We consider a secure processor within the system unit of a workstation. The secure processor is configured as a processor on the workstation's I/O bus. Logically, the workstation regards the secure processor as a server which happens to be local to the workstation.

7.1.1.1 Software

For this example, consider a PS/2 machine running the OS/2 operating system. The workstation is fitted with a secure coprocessor, which has performance roughly equivalent to that of the workstation's main processor. The coprocessor is also instruction-set compatible with the workstation's main processor. As described in "Secure Processor Software Architecture" on page 19, the secure coprocessor is securely loaded with an operating system layer that mimics a subset of the OS/2 functional interface provided to applications.

The secure processor has hardware support for cryptography. The secure processor software includes comprehensive cryptographic support which uses the cryptographic hardware. It is intended to support a wide range of commercial applications of cryptography, and to support the associated key management.

Applications and system functions that are considered sufficiently important may be chosen for execution in the secure processor, which can provide cryptographic guarantees of integrity of the application, and cryptographic guarantees of the secrecy of the application and its data. The user might not make the choices of which applications should be run in the secure processor; these choices may be administrative decisions.

In this example, the coprocessor functions as a secure server. System functions that require guarantees of physical security, for example those that use cryptography and must keep cryptographic keys or plaintext physically secure, would be run on the secure processor. Client processes running on the main processor could make functional requests to the server processes running in the secure coprocessor. For example, the client-server interface could be a remote procedure call mechanism. A client could make a remote procedure call to an application service executing in the secure processor.

The system microcode is loaded into persistent storage. It need only be loaded if it must be updated or if the cryptographic keys associated with it need to be changed for some reason. The system microcode layer is loaded as plaintext by a responsible administrative person. The load is done from a secure workstation, in a trusted location. This is because cryptographic keys are loaded along with the system microcode layer. These keys used to assure secure loading of the system services layer, and must not be compromised.

The system services layer is loaded into the secure processor's RAM via communication with the workstation's main processor. This layer is loaded when the workstation itself is powered up. It is securely loaded from the workstation's disk, where it is stored encrypted. The keys loaded along with the system microcode are used to decrypt the system services layer, and to verify its integrity. When this layer has been decrypted and its integrity verified, it is given control.

In like manner, the operating system layer is securely loaded. Like the system services layer, the operating system layer is resident in the secure processor during normal operation. To applications, the operating system layer closely resembles OS/2. Tools used to develop OS/2 applications can be largely used to develop secure processor applications. The constraints are that the OS/2 emulation is not complete (some function is excluded for security reasons) and there is an additional development step involving encryption of the application.[3]

Secure processor applications prepared on another workstation have their associated cryptographic keys and checksums transmitted via a cryptographically secured communication path to any secure processor in which they will be run.

7.1.1.2 Hardware

For this example the target system will be a PS/2 Model 80 or equivalent micro-channel based system. The secure processor's performance is intended to match that of the target system. To permit the execution of arbitrary applications in the secure processor, it will execute the target system's native instruction set. The secure processor should take advantage of all available system features which improve system performance and ease interface requirements.

A present example of a workstation secure processor is the Transaction Security System [IBM91]. It incorporates a secure processor on a card in both PC and PS/2 versions. The example presented here attempts to go beyond the Transaction Security System and is presented as a continuation of development.

Figure 5 on page 30 shows the functions which will be required in a full implementation of a secure processor. Each of the functions will now be mapped into hardware.

The processor needs to execute the Intel 80386 instruction set. Therefore, the processor can be either an 80386 or 80386SX (the choice between the two can be made by cost/performance requirements).

The main storage will have to be large enough to contain the operating system layer as well as applications. The secure processor can utilize page and segment swapping to achieve virtual memory so that the real storage does not have to be large enough to contain all of the programs at once, but there should be sufficient storage to prevent page thrashing. The minimum required main storage size will be in the 4 megabyte range.

The presence or absence of main storage caching is a cost/performance issue. Unless very slow non-interleaved main storage is utilized, caching can be omitted.

The ROS will need to be in the 32 to 64 K byte range and the persistent storage (EEPROM) will need to be in the 64 to 128 K byte range to accommodate the bootstrap and microcode layers respectively.

The secure data storage will need to be in the 16 to 32 K byte range to store the keys and other secret data. As the secure data storage will need to be battery-backed, it should be CMOS static RAM.

The cryptographic device should be one of the commercially available DES chips made by AMD or Western Digital. These are currently the best available for general use at reasonable cost. The cryptographic rate attainable by these devices is in the 1.5 t o 1.8 M byte/sec rate.

The real time clock can be any low power device which can be powered by a battery when the system is shut down. There are a number of CMOS clock chips on the market which meet this requirement.

The primary bus interface will be a microchannel bus master. This will permit the secure processor to perform its transactions with the highest efficiency and will allow the secure processor to communicate directly with the other devices in the system. The secondary interface will be a general purpose 32 bit interface. This interface can be connected to DASD or LAN control

[3] Application encryption and cryptographic checksum computation might be done within the secure processor, so that the encryption keys are not exposed. However, the application's integrity before encryption must also be ensured, so in any event such applications should be prepared on highly trusted systems.

circuitry. This interface can also be connected to other devices or can be omitted for cost considerations.

The following table outlines the devices used in secure processor implementation for a PS/2 Model 80 microchannel system.

Name	Part	Comment
Processor	80386	386SX
Main Storage	DRAM	1 M Bytes
Read Only Storage	ROM	64 K bytes
Persistent Storage	EEPROM	64 K bytes
Secure Data Storage	CMOS RAM	64 K bytes Battery-backed
Cryptographic Device	DES Chip	Very fast
Real Time Clock	CMOS Clk	Battery-backed
Primary Interface	MCA	Currently Bus Slave
Secondary Interface	16 Bit	General purpose

Figure 9. Secure Processor Implementation Table

7.1.1.3 Physical Security

Workstations are often placed in areas of low environmental security. If high value data (i.e. payroll, accounting, personnel) is to be placed on these systems, physical security for the secure processor is a necessity.

To determine what kind of physical security is required in a given situation the system must be evaluated. Evaluation consists of determining the environmental security and the value of the data. With this information one can specify a physical security system which will be appropriate for the conditions [WEIN88].

In this example, the environment is a standard office and the data is valuable. Minimum protection for the secure processor would include a means for detecting tampering and responding to intrusion [WEIN87], or a means for ensuring that intrusion leaves evidence which can be audited at a later time. In addition, some means of tamper resistance is useful for slowing down any attempt to gain entrance to the interior of the secure processor.

Means exist to deter probing of electronics by sophisticated and determined intruders. It is not possible to go into detail on this methodology because many of the techniques are proprietary. When developing a secure processor, as described here, the techniques can be implemented to a level sufficient to protect the system as required.

7.1.2 Mainframes

7.1.2.1 Software

For mainframes of the 370 class, there are several dramatically different secure processor implementation strategies. A secure processor could be a separate tightly coupled high performance 370 security processor. This option is unlikely to be viable. A secure processor could also be channel attached, and not of comparable performance to the main processor.

The channel-attached secure processor could be securely loaded with an operating environment, much in the manner that a secure processor for a workstation is loaded, as described in "Software" on page 46. Applications would be securely loaded into it in a similar manner.

A third possible implementation strategy is to implement a secure processor in a virtual machine. PR/SM [IBM88], a newly announced product, could be used to implement a secure processor with very high logical security. PR/SM could be used to provide secure processor software with its own private operating environment, with hardware-enforced constraints on communications with other operating environments. IBM's VM operating system could be used in a similar manner to create secure server machines, although the level of logical assurance would be much lower than with PR/SM. In IBM VM systems, many security functions are in fact given a devoted virtual machine.

The virtual machine approach is unusual in that the virtual machine has exactly the same physical security characteristics as its client virtual machines, because they are in fact all the same machine. Environmental security must be substituted. Logical security may also be harder to demonstrate, because of the lack of physical separateness. Without physical separateness, it is harder to prove that there are no undocumented communications paths.

In a mainframe environment, data volumes are typically very high, so if support for bulk encryption is desired, hardware cryptographic support is a necessity.

7.1.2.2 Hardware

Secure processors can be implemented for mainframe systems in a variety of ways. If performance is not critical, the workstation described above can be attached to the mainframe as a server. This is currently done using the IBM 4753 Network Security Processor and the IBM 4755 Cryptographic Adapter. This is sufficient for user authentication services and similar applications. If bulk encryption of data or communications is required, the performance of the secure processor will need to be substantially greater.

A possible approach to implementing a secure processor on a mainframe system is to use a multiple environment system like PR/SM[IBM88]. PR/SM is a newly announced IBM product which allows a single mainframe system to be divided into multiple, separated environments. This would allow the secure processor functions to be emulated in software and remain isolated from other tasks on the system.

A secure processor could, of course be designed and implemented for a mainframe system. If this technology becomes pervasive that is likely to happen. Performance would improve dramatically if the secure processor was a mainframe processor.

At some point between software emulation and full hardware implementation, a hardware cryptographic engine running at mainframe speeds would be useful. This would permit a significant performance increase over pure software emulation without all of the complexity and cost of a full hardware implementation.

Any of the above implementations would be able to implement or emulate the complete secure processor as described in "Secure Processor Hardware Architecture" on page 29.

7.1.2.3 Physical Security

Mainframes, unlike most workstations, are usually kept in a more secure environment. However mainframe systems can store a great deal more valuable data than a workstation. In installations where the system is in an environment that is sufficiently secure to protect the value of the data, no additional physical security is needed. However, if the service people and building maintenance employees who have access to the system are not fully trusted, physical security will still be necessary.

If a workstation is being used as a server, the physical security can be the same as described previously. If an approach like PR/SM or equivalent is used, and the environment was not adequately secured, physical security may have to be employed on the system covers or at the card/board level. This could be very difficult and expensive and would probably be the least desirable situation to have to physically secure. Mainframe computers are large diffuse systems that have processing spread over a large physical area. Environmental security is still the best approach. However the base level of physical security which is a result of the system's sheer complexity is still significant and increases the overall security of the system.

7.1.3 Smart Cards

In this section, we consider a secure processor implemented as a smart card. A smart card is a device that resembles a credit card. It contains a microprocessor and other components, and has an I/O connection that allows it to communicate with external devices.

7.1.3.1 Software

Smart cards currently have storage and performance constraints which force simplifications of the software architecture. In particular, smart cards are likely to be fixed function, with all their software embedded in ROM. A tenable alternative is to put a bootstrap layer in ROM and use the bootstrap layer to load microcode into an EPROM or EEPROM.

Consider a smart card with an 8051, some EPROM for the microcode, some RAM, and some EEPROM for secure data storage and loaded microcode. Having more layers of software than a bootstrap layer and a microcode layer makes little sense because the microcode will likely be installed once, as plaintext microcode, by the manufacturer or customer. The functionality of the smart card, what amounts to small fixed application base, is installed in the card's EEPROM.

Cryptographic services are likely to be a major part of the function of a smart card. For many smart card applications, implementing the cryptographic services in microcode is reasonable. There is typically a time constraint; most transactions should take no more than a few seconds. Common key algorithms such as DES can be implemented in microcode on an 8051 with acceptable speed. Public key algorithms such as the RSA algorithm currently have performance problems when implemented in software on a smart card.

7.1.3.2 Hardware

Smart card implementations of secure processors will generally represent useful subsets of secure processors. Smart cards have smaller quantities of available storage of all types and generally do not have the availability of battery backed RAM for secure data storage. Despite these limitations smart cards are useful for authorization processes and other such applications where size and processing power are not the primary considerations.

A typical example of the processor used in a smart card is the Intel 8051. This processor typically contains 4 k bytes of EPROM and 128 bytes of RAM. If this is not sufficient storage to hold the bootstrap, IML, and system services layers, external EEPROM has to be added. The same applies to the RAM.

EEPROM would have to be added to to serve as secure data storage, however care must be taken in the physical layout on the device so that the lines to the EEPROM are not easily accessible to probing.

A smart card implementation will by necessity be a secure processor subset. The following is a mapping of secure processor function into a smart card sized system. Referring to Figure 5 on page 30, the processor will be an 8051 or equivalent type processor. Persistent storage and ROS may be put together into a single EPROM or the persistent storage may be moved to EEPROM to allow reloading of code. Primary storage will be limited to less that 256 bytes in many cases and will be used to store dynamic operating data which will be lost on power down. Arbitrary loading of user programs is not envisioned in this type of implementation.

Secure data storage will have to be accomplished using EEPROM rather than RAM owing to the lack of battery back-up in current designs. The lack of battery back-up will also necessitate the omission of a real-time clock.

Cryptographic services will most likely have to be accomplished in software.

The primary interface will be a bit serial interface, assuming that the ISO interface standard is to be maintained. There will probably not be a secondary interface unless a display and keypad are added.

Smart cards are an emerging technology. It is not unreasonable to assume that in the near future many of the limitations of smart cards such as lack of storage and absence of battery capability will

be alleviated. This example is based upon commonly available technology. The following table outlines the functions for a typical 8051 type smart card.

Function	Part	Comment
Processor	8051	Microcontroller
Main Storage	RAM	1/4 - 1 K Bytes
Read Only Storage	ROM	2 to 4 K bytes
Persistent Storage	EEPROM	2 to 16 K bytes
Secure Data Storage	EEPROM	1/4 - 2 K bytes
Cryptographic Service	DES	In software
Primary Interface	Serial	ISO standard
Secondary Interface	Optional	Display & keypad

Figure 10. Smart Card Secure Processor Implementation Table

7.1.3.3 Physical Security

Smart cards have a significant amount of "built in " physical security. They typically use a single integrated circuit. Probing an integrated circuit directly is a complicated task requiring significant skills and specialized equipment.

Under normal circumstances, the only major concern for the physical security of smart cards is preventing probing of inter-chip connections in multi-chip designs. In many cases inter-chip connections are relatively simple to probe, requiring only common tools and a steady hand. Single chip implementations do not suffer this weakness.

Multi-chip implementations require protection for inter-chip connections. The most straightforward method of accomplishing this is to encrypt any data which leaves the processor I.C. This would include data being stored in a remote EEPROM used for secure data storage, etc.

In applications where very high value data is contained in a smart card there are methods of improving the physical security of integrated circuits. These methods go beyond the scope of this document.

7.2 In a Distributed System

We now consider how a distributed system using secure processors might work. The intent of this section is to show how an actual system might be installed, used, and administered.

7.2.1 Configuration

Widgco is a small widget manufacturer. Its headquarters are in Florida, and its single manufacturing site is in Montana. The headquarters DP staff maintains a small mainframe in a "glass house," which is connected to a token ring LAN supporting a number of workstations. The manufacturing site has a minicomputer on the factory floor for controlling the manufacturing line. It is connected to another token ring LAN supporting several workstations. The LANs at the two sites are connected via a satellite link, keeping them in communication at all times.

Widgco has implemented a distributed computing system, in which clients send requests to servers for various services. Any machine may be a client, but Widgco has wisely restricted the servers according to their ability to support their function securely.

The security administrator's workstation is kept locked in her office, and is disabled whenever the administrator is not present. Various other workstations are used as both normal workstations and as servers. They reside in the offices of employees. These offices are not usually locked. To ensure

the security of their databases, each server is equipped with a secure processor option, as described in "Workstations" on page 46.

7.2.2 *Installation*

Suppose that Widgco wants to set up a new workstation server to maintain its personnel database. The security administrator must identify the new workstation, and its security characteristics, to the current distributed system.

7.2.2.1 Security Policy

Widgco's distributed system security policy states that "sensitive" programs and data must either be (a) encrypted with DES, or (b) reside in plaintext in physically/environmentally secure areas. The "glass house" and the security administrator's office are considered environmentally secure, so the mainframe and the security administrator's workstation are secure. In addition, secure processors within workstations in employee's offices are considered sufficiently secure to handle sensitive programs and data. The token rings, and the satellite link, are not considered sufficiently secure to permit sensitive plaintext programs or data at any time.

"Sensitive" includes business plan information, financial transactions (EFT, etc.), user authentication to the system, and personnel information. It also includes the basic programs that handle this data. Most of the data kept on the systems consists of memos, electronic mail, personal calendars, and so on. These are not usually considered sensitive.

7.2.2.2 Security Characteristics of the New Server

The new server is a workstation with a secure processor option installed inside of it. The server functions will be handled by the secure processor, rather than by the main workstation processor. The workstation itself will be located in an employee's office, which will not necessarily be locked or attended all the time.

The personnel data server will be located in Widgco's Florida offices, but their Montana site will occasionally need access to the information. Thus, sensitive information will have to be transmitted on both LANs, and on the satellite link between them. Widgco's security policy requires these transmissions to be encrypted.

It is the responsibility of the software inside of the secure processor to enforce this security policy. It must encrypt all sensitive information destined for the LAN. The information must also be encrypted if it is to be stored on any mass storage device that may be left unattended, like a floppy disk or a hard disk.

Sensitive information may be left in plaintext within the secure processor, of course. It may also be given in plaintext to the main processor of the workstation if the secure processor can guarantee that the user that is currently logged on is authorized to have that information.

7.2.2.3 Installing the New Server

Before installing the new server workstation on the network, the security administrator must initialize its secure processor. The secure processor's bootstrap code layer is used to load the microcode layer into the secure processor's persistent storage. This microcode layer contains one or more cryptographic keys and manipulation detection codes. These determine which system services layers can be loaded onto the secure processor when it is powered up. Since the cryptographic information that is loaded with the microcode layer ends up determining the capabilities of this secure processor, it is very important that this initialization take place in a secure location.

When the new server workstation is installed on the network, the security administrator must tell the system about its security characteristics. The system should know the network port it will use, so that messages can be routed correctly, of course. It should also know that there are two processors on this network port: one secure processor and one normal workstation processor. These have different physical security characteristics, so they are trusted with sensitive information to different extents. Once this information is determined, the security administrator can enter it into

a database, so that applications can query the security characteristics of various machines. This database may be centrally administered, or it may be distributed.

7.2.3 *Daily Use*

To the users, Widgco's distributed system appears much like the systems of today. In the morning, the users log onto the system via their workstations. In many cases, they will simply provide userids and passwords, which their workstation will forward to an authentication server in order to authenticate the user securely. This authentication server may be located in any appropriately secure machine. It may be a program running on the mainframe in the "glass house." Or, it may run on a secure processor within a more exposed workstation. It may even run on the secure processor within the user's own workstation.

Users who handle particularly sensitive information, such as the mainframe operators and the personnel coordinator, may need to identify themselves more securely. They may use a smart card, or perhaps a biometric device such as a signature verification pen. These devices can provide a physically trusted path to a secure processor via a directly-attached cable. Or, they can provide a logically trusted path to a secure processor by encrypting their communications with the secure processor with a key known only to the device and the processor.

Once logged on, the user can use non-sensitive programs and data normally. The programs may run in the office workstations. The data may be transmitted in plaintext over the various networks, and stored in plaintext on mass storage media.

When they appear in plaintext, sensitive programs and data will have to be restricted to machines with sufficient environmental and physical security. They must be encrypted everywhere else. These restrictions are enforced by the software running in the secure machines, though, and the users may not even be aware that it is happening.

Suppose that a request for personnel data is made from an authorized user at a client machine in the Montana site to the personnel data server in the Florida site. The client needs to verify its identity to the server in such a way that it cannot be compromised en route. There are a variety of cryptographic protocols that will ensure this. The server machine may wish to verify that the client machine is registered as having appropriate environmental and physical security to deal with the information once it has been received. The server may obtain the security characteristics of the client from the database machine set up to handle this data.

Once the client has been verified as sufficiently secure, and the user has been verified as authorized to receive the information, the personnel data server may respond to the request. In doing so, it must ensure that the information is encrypted, and that the client machine is capable of decrypting it.

7.2.4 *Administration*

Administration of this system is much like it is for any distributed system. Normal procedures may be used for updating the list of authorized system users, updating the access control database, and auditing. The only differences appear because of the physical and cryptographic security that has been introduced.

Some physical security systems will cause sensitive information within the secure processor, such as cryptographic keys, to be erased in the event of tampering. This is to prevent the compromise of this information by hardware means. Since this information is vital to the correct operation of the system, it should be backed up securely. This could be done by having the security administrator keep a written list, locked in a safe, of cryptographic keys for each machine. In a large distributed system, however, this is impractical. Fortunately, there are cryptographic protocols for creating and installing backups securely. The backup files are themselves encrypted, and can be stored on conventional mass storage media.

It is common to use debuggers of various sorts to identify and correct difficult system problems. These may permit the user to read and write directly to a disk, for instance, bypassing the safeguards in the operating system. Or, they may permit the capturing and injection of packets on networks, enabling the user to read and write arbitrarily on a network.

Clearly, these abilities must be very carefully controlled in a secure system. The architecture we describe includes the ability to install manipulation detection codes (MDCs) on a secure processor. The set of applications that may be run on a processor can be limited to those for which a valid MDC is present. Control of debugging utilities can be as simple as not including debuggers in the list of programs that secure processors may run, and ensuring that sensitive information is encrypted. For those situations in which it is necessary to examine or change sensitive information, special utilities can be used. The ability to run these utilities would require the security administrator's authorization, and their use would typically be authorized only for the duration of the problem.

7.3 Conclusion

The above examples are intended to show a few ways in which this architecture could be implemented and used. The intent of this section has been to give the reader a more concrete idea of how secure processor would be used in practice. The architecture has been designed to permit a wide variety of possible implementations, depending upon the price, performance, and function requirement in each case. There are certainly many more possibilities than have been discussed in this section.

8.0 Conclusion

The Citadel architecture addresses the problems of protecting information assets in physically exposed environments. These environments are becoming more common computing environments as the trend towards small computers and distributed processing continues. In this document, we have introduced several parts of the Citadel architecture.

We have described the use of physical, environmental, and cryptographic security to protect valuable information assets from physical attack. Object-oriented encryption was introduced as a way to use these aspects of protection in a unified security policy.

Secure processors are necessary components of systems that are secure against physical attack. Even the most critical information must appear in plaintext within a computing system at some time. Secure processors permit valuable plaintext information to be processed securely, even in physically exposed environments.

Models of single and distributed systems point to the roles that secure processors can play in protecting systems. We have examined several of these roles. As secure parts of a secure distributed system, they are ideal places in which to implement reference monitors. When they interact with less secure parts of the system, they are best thought of as secure servers.

We have explored architectures for secure processors that can span the range from hand-held systems to mainframes. The details of secure processor implementations vary over this range of systems, but their basic characteristics remain the same

The software architecture of secure processors permits all of their functions to be customized and updated by security administrators. This includes their cryptographic architecture, the details of their security kernel, the operating system that is emulated by the secure processor, and the application software that can be used.

The hardware architecture of secure processors is designed to ensure the security of information within them. It also delivers cryptographic services to programs running within the secure processor, and to the system in which it is placed. Several alternatives for the location of secure processors in systems were examined. Each has advantages. The optimal placement will depend upon the system in which the secure processor is being placed, and on the specific role it will play in that system.

We have given an overview of physical security. Included was an introduction to the kinds of attacks that are possible, and the defensive techniques that are available. These are important to any security policy that deals with the possibility of physical attacks.

We are continuing to refine and develop this architecture. In particular, we will address several aspects in more detail in companion documents. The first is a physical security evaluation system. This is a detailed methodology for rating physical security systems, and determining their suitability for various uses. The second is a detailed functional architecture for secure processors. This describes the functions that secure processors can implement. It covers specific cryptographic techniques for loading the software layers. It also details the administrative functions necessary to update authorizations, create backups, and so on. Last is a more detailed explanation of object-oriented encryption, and its role in implementing a uniform security policy in distributed systems.

Glossary

Availability: That aspect of security that deals with the timely delivery of information and services to the user. An attack on availability would seek to sever network connections, tie up account or systems, etc.

Asset: Anything which is valuable to an individual or organization. (See "Information Asset.")

Compromised: Exposed to illicit disclosure or modification. This is the state of a system after a successful attack on it. A system which is "Untrusted" (see below) is not compromised until it has been attacked successfully.

DES: Data Encryption Standard. A symmetric cryptosystem, using a 56-bit key and 64-bit blocks of data, which is widely used in commercial applications.

Environmental Security: That aspect of the system security which deters or prevents an attacker from gaining physical access to the system. Locked rooms, motion detectors, and guards are aspects of environmental security.

Information Asset: A piece of information which is sensitive or valuable to an individual or organization. This can be simple data, like a document describing a competitive strategy or a bank balance. It can be more abstract data, such as a cryptographic key, a password, or an access control database. It can be a program, such as an electronic funds transfer program or an auditing program.

Integrity: That aspect of security that deals with the correctness of information or its processing. An attack on integrity would seek to change an account balance illicitly, improperly alter the access control permissions for a user, falsify audit records, etc.

Logical Security: That aspect of system security which ensures the security of a system, even though an attacker can perform logical input and output operations on it. Logical security encompasses security kernels, access control systems, auditing facilities, logon programs, etc.

Logical Trojan Horse: Any program designed to do things that the user of the program did not intend to do. An example of this would be a program which simulates the logon sequence for a computer and, rather than logging the user on, simply records the user's userid and password in a file for later collection. Rather than logging the user on (which the user intended), it steals the user's password so that the Trojan Horse's designer can log on as the user (which the user did not intend).

Logically Trusted Path: A communications path, usually between a user and a security kernel, which is not subject to tampering via software.

Orange Book: An informal name for the Department of Defense Trusted Computer System Evaluation Criteria [NCSC85]. The name stems from the color of the book's cover.

Physical Security: That aspect of system security which deters or prevents an attacker from illicitly viewing or modifying information assets, even if the attacker has physical access to the system. Physical security systems are divided into tamper resistant systems, tamper evident systems, and tamper responding systems.

Physical Trojan Horse: Any piece of hardware designed to do things that the user of the hardware did not intend to do. An example of this would be a hardware module that captures the user's keystrokes while the user is logging on to a system. Rather than just passing the keystrokes to the logon program (which the user intended), it steals the user's password so that the Trojan Horse's designer can later log on as the user (which the user did not intend).

Physically Trusted Path: A communications path, usually between a user and a secure processor, which is not subject to tampering via hardware.

Reference Monitor: An entity within a secure computing system that mediates all accesses by systems subjects (users, processes, etc.) to system objects (files, communications media, etc.).

RSA: A public-key cryptosystem, named after its inventors (Rivest, Shamir, and Adelman). It is based on exponentiation in a finite field, and is currently the most widely used public-key cryptosystem.

Secrecy: That aspect of security that deals with the restriction of information to those who are authorized to use it. An attack on secrecy would seek to view cryptographic keys, discover passwords, read files, etc., to which the attacker was not entitled.

Secure Enclosure: An enclosure whose contents are protected by a physical security system. A secure processor will typically be housed in a secure enclosure.

Secure Processor: A computing system which has adequate physical and environmental security to process information assets of a given value. A secure processor usually processes high-value information assets. It usually has significant physical security, to compensate for inadequate environmental security.

Security: When applied to computing systems, this denotes the authorized, correct, timely performance of computing tasks. It encompasses the areas of secrecy, integrity and availability.

Security Kernel: The implementation of a reference monitor (See "Reference Monitor") on a real system. The security kernel provides the base security functions for an operating system, which allow it to control the access of each system subject to each system object.

Trojan Horse: Any component of a computing system, whether software or hardware, designed to do things that the user of the component did not intend for it to do. (See "Logical Trojan Horse" and "Physical Trojan Horse.")

Trusted: Having a sufficient assurance of security so as to be capable of handling valuable information assets without their being compromised.

Trusted Computing Base: The totality of protection mechanisms within a computer system (including hardware, firmware, and software) the combination of which is responsible for enforcing a unified security policy.

Trusted Path: A communications path, usually between a user and a security kernel, which is not subject to tampering.

Uncompromised: Secure from illicit disclosure or modification. This is the state of a system before any attack on it has been successful. Uncompromised systems, whether "Trusted" or "Untrusted," operate securely.

Untrusted: Lacking a sufficient assurance of security to be capable of handling valuable information assets without their being compromised.

Bibliography

[ANDE72] "Computer Security Technology Planning Study", ESD-TR-73-51, Vols. I and II, USAF Electronic Systems Div. Bedford, Mass. (Oct. 1972)

[ANSI85] ANSI 9.17-1985 **American National Standard** "Financial Institution Key Management (Wholesale)"

[DENN82] "Cryptography and Data Security", Dorothy Elizabeth Robling Denning, Addison-Wesley, 1982

[IBM72] "The Considerations of Physical Security in a Computer Environment", IBM Corporation, G520-2700-00 (Oct., 1972)

[IBM88] "3090 Processor Complex: Processor Resource / Systems Manager Planning Guide", IBM Corporation, GA22-7123-0 (May, 1988)

[IBM90a] "Common Cryptographic Architecture: Cryptographic Application Programming Interface Reference", IBM Corporation, SC40-1675 (Nov, 1990)

[IBM90b] "ICSF/MVS General Information", IBM Corporation, GC23-0093 (Aug, 1990)

[IBM91] "Transaction Security System: General Information and Planning Guide", IBM Corporation, GA34-2137 (Mar, 1991)

[JUEN85] "Message Authentication", R. R. Jueneman, S. M. Matyas, C. H. Meyer, IEEE Communications, Vol. 23, No. 9 (Sept. 1985)

[KENT80] "Protecting Externally Supplied Software in Small Computers", Stephen Thomas Kent, PHD Thesis, M.I.T. Laboratory for Computer Science (September 1980)

[MEYE82] "Cryptography: A New Dimension in Computer Data Security", Carl H. Meyer and Stephen M. Matyas, John Wiley and Sons, Inc, 1982, pp. 351

[NCSC85] "Department of Defense Trusted Computer System Evaluation Criteria", DOD 5200.28-STD, National Computer Security Center (1985)

[NCSC87a] "Trusted Network Interpretation", NCSC-TG-05, Library No. S228,526, Version 1, National Computer Security Center, (July 31, 1987)

[NCSC87b] "Sub-System Evaluation Report, Micronyx. Inc. Triad Plus Version 1.3", National Computer Security Center, CSC-EPL-87/006, Library No. S228,557 (August 14, 1987)

[NESS87] "Factors Affecting Distributed System Security," Dan M. Nesset, IEEE Trans. on Software Engineering, Vol. SE-13, No. 2 (Feb. 1987), pp. 233-248

[PIET87] "The security kernel: background and elements," Security, Vol. 9, No. 3 (1987) pp. 131-138

[SAYD89] "LOCK trek: Navigating Uncharted Space", O. Sami Saydjari, Joseph Beckman, Jeffrey R. Leaman, 1989 IEEE Symposium on Security and Privacy

[STEI88] "Kerberos: An Authentication Service for Open Network Systems", Jennifer G. Steiner, Clifford Neuman, Jeffrey I. Schiller, USENIX Winter Conference, February 9-12, 1988

[WEIN87] "Physical Security for the μABYSS System", Steve H. Weingart, 1987 IEEE Symp. on Security and Privacy

[WEIN88] "A Physical Security Evaluation System", Steve H. Weingart, Steve R. White, Glen P. Double, William C. Arnold, 1988

[WHIT87] "ABYSS: A Trusted Architecture for Software Protection", Steve R. White and Liam Comerford, 1987 IEEE Symp. on Security and Privacy

[ZIMM80] "OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection", Hubert Zimmermann, IEEE Transactions on Communications, Vol. Com-28, No.4, pp.425-432 (April 1980)