1992

A Distributed Security Architecture for Large Scale Systems

Shepherd, Simon John

http://hdl.handle.net/10026.1/2143

http://dx.doi.org/10.24382/4007
University of Plymouth

A Distributed Security Architecture for Large Scale Systems

Simon John Shepherd
B.Eng(Hons)

A thesis submitted in partial
fulfillment of the requirements of the
Council for National Academic Awards
for the degree of
Doctor of Philosophy

Sponsoring Establishment:
Network Research Group
Faculty of Technology
University of Plymouth, UK.

Collaborating Establishment:
IBM (United Kingdom) Laboratories
Hursley Park, Winchester, UK.

June 1992

A Distributed Security Architecture for Large Scale Systems

Simon John Shepherd, *B.Eng(Hons)*

Abstract

This thesis describes the research leading from the conception, through development, to the practical implementation of a comprehensive security architecture for use within, and as a value-added enhancement to, the ISO Open Systems Interconnection (OSI) model.

The *Comprehensive Security System* (CSS) is arranged basically as an Application Layer service but can allow any of the ISO recommended security facilities to be provided at any layer of the model. It is suitable as an 'add-on' service to existing arrangements or can be fully integrated into new applications. For large scale, distributed processing operations, a network of *security management centres* (SMCs) is suggested, that can help to ensure that system misuse is minimised, and that flexible operation is provided in an efficient manner.

The background to the OSI standards are covered in detail, followed by an introduction to security in *open systems*. A survey of existing techniques in formal analysis and verification is then presented. The architecture of the CSS is described in terms of a conceptual model using *agents* and *protocols*, followed by an extension of the CSS concept to a large scale network controlled by SMCs.

A new approach to formal security analysis is described which is based on two main methodologies. Firstly, every function within the system is built from layers of provably secure sequences of finite state machines, using a recursive function to monitor and constrain the system to the desired state at all times. Secondly, the correctness of the protocols generated by the sequences to exchange security information and control data between agents in a distributed environment, is analysed in terms of a modified temporal Hoare logic. This is based on ideas concerning the validity of beliefs about the global state of a system as a result of actions performed by entities within the system, including the notion of *timeliness*.

The two fundamental problems in number theory upon which the assumptions about the security of the finite state machine model rest are described, together with a comprehensive survey of the very latest progress in this area. Having assumed that the two problems will remain computationally intractable in the foreseeable future, the method is then applied to the formal analysis of some of the components of the Comprehensive Security System.

A practical implementation of the CSS has been achieved as a demonstration system for a network of IBM Personal Computers connected via an Ethernet LAN, which fully meets the aims and objectives set out in Chapter 1. This implementation is described, and finally some comments are made on the possible future of research into security aspects of distributed systems.

CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

Firstly, I would like to thank my supervisors; Peter Sanders, Director of Studies, for his unfailing guidance, tolerance and concern; and Dr Colin Stockel for his unerring judgement in all things academic.

I also thank the IBM (United Kingdom) Laboratories, in particular Peter Smith, for their financial support during much of this research. It would not have been possible without them.

I would also like to extend my appreciation to Prof Dr Seàd Mùftfc of the University of Sarajevo in what was Yugoslavia, to Dr Ahmed Patel of University College Dublin, Ireland, to Unto Pulkkinen of the Technical Research Centre of Finland in Helsinki, and to Peter Van Eetvelt for their helpful advice. I also thank Prof Peter Watson of the University of Bradford for giving me time to complete this thesis after my taking up a lectureship at the University.

And by no means least, I am deeply grateful to my wife for her unfailing support and encouragement, and her determination for this research to be brought to a successful conclusion.

DECLARATION

The author hereby certifies that the material contained herein is the sole work of the author, who is also responsible for the computer programs, results, the interpretation thereof, and the conclusions drawn; and that this material has not been submitted for any previous academic award.

[illegible]

Simon J Shepherd
June 1992

First, my fear; then, my curtsy; last, my speech. My fear is, your displeasure; my curtsy, my duty; and my speech, to beg your pardons. If you look for a good speech now, you undo me; for what I have to say is of mine own making; and what indeed I should say will, I doubt, prove mine own marring. But to the purpose, and so to the venture.

William Shakespeare
King Henry IV, Part II

Only for you, children of doctrine and learning, have we written this work. Examine this book, ponder the meaning we have dispersed in various places and gathered again; what we have concealed in one place we have disclosed in another, that it may be understood by your wisdom.

Heinrich Cornelius Agrippa
von Nettesheim
De occulta philosophia, 3, 65.

1. INTRODUCTION

1.1. Network Security

The joining together of computer networks is widespread. The financial community make use of international Wide Area Networks, large corporations network their plants together and higher education uses networks for collaboration in research. The military uses networks to coordinate operations and for communications at all levels of security. Offices use Local Area Networks to decrease the use of paper in the office and to facilitate sharing of expensive peripherals [COOP,1988]. As a result, the use of networks continues to grow and the traffic across networks is increasing all the time.

As the use of networks increases, the amount of sensitive data carried increases. Network users, particularly the military, financial and commercial users, need to be able to protect their networks from infiltration and abuse. Attackers who tap network lines and read, modify or destroy network traffic must be stopped [STOL,1988]. Due to the long communications distances involved in most networks, it is almost impossible to protect the physical medium being used to carry the sensitive data. It is usually possible for the attacker to gain access to the medium, and since this cannot easily be protected, the data must be protected instead. The advent of optical fibre technology for network communications will go some way to alleviating the problem of tapping, because it is very difficult to place a tap on a fibre optic cable. This technology does nothing, however, to overcome the problem of *denial of service* attacks (see section 2.3.) due to physical damage to, or destruction of, the cable.

The subject of network security is a large one and there are many approaches to security provision, each of which need to be considered in order to provide a secure network.

Open Distributed Processing (ODP) is the conceptual framework within which systems of diverse origin, application and location can interact freely if required (see Chapter 3). Because there may be many different components, operations, resources and entities involved in such an arrangement, a network constructed within this framework presents a very convenient target for various attacks and illegal operations, which means that protection of the system resources and assets is becoming an increasingly important factor in network design.

Systems currently exist which have made some attempt to implement security measures. In many cases, the most effective are those which were conceived from the outset to offer security as a prime function and are typical of those used by governments, military and financial institutions.

The majority of other communications systems which exist, however, were not originally conceived with the security function in mind and make no provision for it other than allowing the execution of specific applications which have security measures built into their facilities on an individual basis. A typical example is the JANET academic network which does not provide encryption facilities, but may be used to send encrypted messages if the appropriate mechanisms are provided by individual system users. A secure E-mail facility using the RSA algorithm has been implemented in the Network Research Group at the University of Plymouth and is used for secure correspondence with collaborators at home and abroad. The disadvantage of this approach is that it is very difficult to assess the overall strength of such a system, where security is provided on an individual 'ad hoc' basis, due to the absence of a formal architecture capable of rigorous analysis.

Information security technology encompasses all measures that are used to protect information from unauthorised disclosure, modification or destruction. In an age where information is widely regarded as a valuable commodity, information security has become particularly vital.

The technology of information security has evolved through three basic stages: data protection, system protection and system verification.

Encryption has been used since antiquity to protect information from unauthorised disclosure. It was the advent of high-speed digital computers, however, that made the wide-scale use of sophisticated encryption systems possible for the first time. The principles and mechanisms of encryption are well understood and many ingenious new algorithms are reported every year. In particular, the advent of public key ciphers [DIFF,1976][RIVE,1978] has allowed the possibility of encryption over large-scale networks without the logistical problems of key distribution associated with classical (symmetric) cipher systems. Given that the fundamental assumptions upon which the security of these ciphers are based are true, the strength of modern ciphers can be taken as being very high indeed.

Security technology, however, must do much more than just protect data. The systems that handle the data must themselves be protected to safeguard their operating principles as well as the data they contain. Before the advent of communications, such protection could be ensured by *physical* means, such as placement of the equipment in locked rooms. As long as the equipment could not be operated remotely, this solved the problem. The advent of communications networks between data processing systems, however, has compounded the problem enormously. If physical protection of the data channels is impossible then ways *logically* to protect the channel must be sought instead.

The main problem, however, in the implementation of a successful security system, lies in the difficulty of verifying that the system will behave *exactly* as it should. Otherwise, it is not possible to give assurances about the absolute level of security offered by the system. It may have undetected design flaws that allow insecure behaviour under certain conditions not foreseen by the designers. It may be that the system could be altered secretly and maliciously by an outsider who wishes to manipulate the system for his own ends. In very simple systems, visual inspection and exhaustive trial is sufficient to verify correctness, but in large, modern systems such an approach is out of the question for the reasons given in Chapter 4. The power of the modern computer system is a double-edged sword: it makes possible very sophisticated encryption, but brings enormous problems in systems verification. It is in this area that *standards* have a very important role to play if these problems are to be overcome, see Chapter 3.

Customers for secure systems have historically been the government, financial and military communities. The government and military requirements fall into three areas: secrecy, integrity, and reliability. Although formal definitions of these terms are given in Chapter 2, these requirements can be summarised respectively as: prevention of disclosure of information to the outside world as well as separating data on a need-to-know basis within the system, protection of data from corruption while in storage or transit and the knowledge that the system and data will always be available when required.

The needs of the financial community are especially demanding. Here, concern for secrecy is equalled and even surpassed by the need for strong data integrity. For example, any part of a financial transaction system must be able to convince not only itself of authenticity, but be able to establish legitimate claims before a judicial authority.

As other commercial organisations see the benefit that they can gain from interconnection of their resources, they too perceive the need for security. Few companies will entrust commercially sensitive information to a communication system which will allow free access to all and sundry.

So important is this problem of security, that the Commission of European Economic Communities (EEC) commissioned a special study into the need for security. It was the largest study of its kind ever undertaken, and was conducted by Coopers and Lybrand [COOP,1988]. The report came to the worrying conclusion that "...computer and network security is inadequate..." and that if standards and systems are not put in place to correct the deficiencies, the problem could "...act as a brake on economic development in the European Community...". The study encompassed seven European countries: the UK, France, Germany, Italy, the Netherlands, Belgium and Sweden. Among the 17 vendors of computer equipment who took part were organisations such as the Digital Equipment Company and British Telecom.

The report discovered that users of networked systems were critically dependent on their systems in 'virtually all aspects of their operational, administrative and financial processes', more so than many believed [RAYM,1988]. In addition, the report found that 'network systems are extensively exposed to disruptive events' and that 'the nature and extent of this exposure is not always understood'.

The conclusion recommends that users adopt 'as a matter of priority, a systematic, methodical, comprehensive and well-founded approach to network system security'. Another important finding of the study from the point of view of this research, is that it is strongly believed that there is '...considerable scope for standardisation in the field of security...'. This could be achieved by rationalising conflicting national legislation and regulations and by developing multi-vendor standards for commonly required security mechanisms. This research addresses the first of these recommendations.

In any discussion of computer security, the central element is the statement of *threat*. The threat is the list of occurrences, both accidental and deliberate, against which the system is to be protected. The study of threat assessment is an art in itself and attempts have been made to address the process from a systems point of view in a rigorous manner [PIER,1988]. Once the threat has been established, a *security policy* (see Chapter 5) is drawn up to address the aspects of the threat and specify the countermeasures to be adopted to minimise the threat. In general, the approach taken is that designers of secure computer systems have little interest in identifying the resources available to specific classes of perpetrators. Instead, they assume that the perpetrator will apply resources in proportion to the value of the protected resource to be subverted.

The concept of a *Comprehensive Security System* (CSS), which can be retrospectively added to an existing data processing system as a value-added function provider and the integrity of which can be demonstrated by formal models and methods of logical analysis, is very attractive to owners and managers of large, existing communication networks. New applications can be written to utilise the security functions on offer, and existing applications can be modified or updated to use the system.

The CSS to be described in Chapter 5 involves the provision of security services for use at, and for the transfer of data between, remote end user entities. Within the ISO model, there are potentially many different services and applications which will benefit from a value-added (an additional feature or service retrospectively added to a system to enhance its usefulness) security system, but in general, each requiring a different combination or sequence of security functions. The CSS is simply another application layer entity. Applications communicate with the CSS via an Application Program Interface (API). Unless applications are written to take advantage of the interface, they cannot avail themselves of security services. The output of the CSS is of a form that is comprehensible to the receiving CSS at the other end of the network, but incomprehensible to any other entity.

1.2. Aim and Objectives of the Research

It is being recognised [COOP,1988] that the use and advantages of information technology will be severely limited within Europe unless sufficient attention is given to all aspects of data security and systems dependability in both the storage and communication spheres [ESPR,1988] [POLI,1988]. Over the last decade, a number of conventional and public key algorithms together with their associated protocols [DIFF,1984] [RIVE,1978] [MERK,1978] have been developed for use in non-military applications, mainly the financial sector, but have operated within relatively small-scale, point-to-point, star arrangements [SUMM,1987] [RUSH,1986] [RUTL,1986] [MUFT,1989]. With the proliferation of large-scale, multi-user networks designed for an open system environment on a global scale, it is becoming essential for suitable data confidentiality, integrity and user identification techniques to be incorporated into a cost effective, simple operating system for the average user [COOP,1988].

The aim of the research program, therefore, was to investigate the possibility of bringing about the realisation of such a system, and comprises the following stages:

1. an initial updating on the latest techniques in the relevant areas of security, many of which are described in [COOP,1988] [DENN,1983] [DTI,1989] [ESPR,1988] [GASS,1988] [MUFT,1989] [MUFT,1990] [MUFT,1992];

2. a detailed examination of the relevant standards in existence [CCIT,1987a] [CCIT,1987b] [DOD,1985] [DTI,1989] [ECM,1987a] [ECM,1987b] [ISO,7498] [ISO,7498-2] [ISO,8571] [ISO,JTC1] [ISO,SC21] [ISO,TC97];

3. a discussion of open system architectures that will be required for the development of the security architecture within the framework of the Open Systems Interconnection (OSI) model [ISO,7498] [ISO,7498-2] [ISO,8571] [ISO,JTC1] [ISO,SC21] [ISO,TC97];

4. an investigation into the potential problems associated with large system operation is reviewed [BLAT,1987] [COOP,1988] [DOD,1985] [ECM,1987b] [ESPR,1988] [GASS,1988] [MUFT,1990] [MUFT,1992] [PATE,1988] [POLI,1988] [RAMA,1990] [RAYM,1988] [RUSH,1986] [RUTL,1986] [SAN,1988b] [SIDE,1988] [SPAF,1988] [STOL,1988] [SUMM,1987] [VOYD,1983]. From this investigation, an overall security policy is developed;

5. a review of complete communication systems from the point of data

generation to its final destination is given. Appropriate security techniques are developed to allow the security policy to meet the needs of the OSI environment;

6. the development of a software demonstration of a secure communications system to run on an Ethernet LAN of IBM Personal Computers;

7. the extension of the above ideas to develop a formal model for the analysis of security operations and mechanisms, to allow the strengths and potential weaknesses to be analysed. The structure and attributes of system entities will be clearly specified and their interactions and constraints specified mathematically;

8. an analysis of the demonstration system using the formal techniques developed earlier;

9. a review of the effectiveness, strengths and weaknesses of the analysis, and recommendations for further development of security systems using this work.

1.3. Layout of Material in the Thesis

This thesis describes the research leading to the conceptual development, practical implementation and formal verification of a Comprehensive Security System. The subject matter is arranged as follows.

The present chapter provides a general introduction to the topic of computer security, discusses with extensive references the motivation, aims and objectives of the research and describes the layout of the material in the thesis.

Chapter 2 covers the background to the concept of security, the definition of and need for security, the concepts of internal and external security, secrecy and integrity and trusted system evaluation criteria. The concepts of users and trust are introduced and defined, and the components of secure systems are discussed. The boundaries of the system in question and various system entities are defined.

Having established the background against which the research is set, Chapter 3 discusses the special requirements of security in *open systems*. The concept of an open system is introduced, and the need for international standards for open systems is discussed, with a status report of current progress and government attitudes to these efforts and current approaches to security in open systems. By far the most important standard to emerge in the field of open systems is the Open Systems Interconnection (OSI) Reference Model drawn up by the International Standards Organisation (ISO). This architecture is the basis for the new research, and the details of the seven-layer model are discussed in detail although it is important to realise that the basic principles of the security methodology may be applied to any open system. By way of example, the important File Transfer, Access and Management (FTAM) service is described to illustrate the principles of the model. ISO themselves have recognised the need for security within open systems, and have drawn up draft recommendations and statements of requirements. Due to the inaccessible nature of the standards documents, the OSI *Security Architecture* is discussed in depth. In order to fulfil the requirements of the recommendations in the standards, it was found necessary to extend the Security Architecture. The requirements for this are outlined together with a discussion of the placement of security provisions within the framework of the Reference Model.

Chapter 4 introduces the concepts underlying existing techniques for the formal analysis and verification of computer systems, including the important state-machine model which is used later in a rigorous manner for the design of the CSS. Methods for controlling the activities of, and data flow between, various system entities are discussed, and existing approaches to methods of decomposition are described. The powerful ideas underlying proofs of correctness via the mechanism of formal logic are introduced. The methods of Hantler and King [HANT,1976] are discussed by way of an example of early attempts at formal logic proofs. The resulting extreme difficulty of conventional debugging techniques when applied to large software systems is examined, and the limitations of existing approaches to formal analysis are then exposed with the need for a new verification methodology being highlighted.

These methods are applied in a new approach to the distributed security problem, the *Comprehensive Security System* (CSS) which is described in detail in Chapter 5. The basic concept of the system is presented and various definitions pertinent to the design are introduced. The design methodology from the conceptual overview, through the generic models, to the agents and protocols which comprise the system is described. The concept of the Security Management Centre (SMC), an important feature of the design, is introduced. The basic implementation for an Ethernet Local Area Network (LAN) of IBM Personal Computers running MS-DOS is described, together with the details of the Application Program Interface (API) to the CSS.

Chapter 6 describes the main body of the research - a new analysis methodology based on a 'bottom-up' design technique for large systems like the CSS described in Chapter 5, but without many of the shortcomings of the analysis and verification methods described in Chapter 4. The fundamental security assumptions upon which the method is founded are described and the layered approach to mechanism generation is detailed. This technique requires that every system function be built from layers of *verified* and *validated* finite-state machines, stored in a library of trusted *functions*. These functions are activated in a *monitored* sequence and a recursive algorithm for the determination and verification of the state of the CSS sequences at any time constrains the number of variables to a point where analysis is feasible. Several examples of security mechanisms generated in this way are given. As well as providing the functions used to manipulate the data from a security point of view (eg encryption), the functions are also used to generate the *protocols* by means of which the *agents* communicate securely. The new methodology therefore includes a modified Hoare logic [HOAR,1969], using the work of Burrows *et al* [BURR,1988], for the verification of protocols from a security standpoint. This is a predicate logic concerning certain formal aspects of *beliefs* about the global system state as a result of *actions* performed on system data by subject entities. The new analysis techniques are then applied to some examples of mechanisms and protocols from the CSS, demonstrating the approach to formal verification.

A review of the findings and limitations of the results and conclusions drawn as a result of it are given in Chapter 7, with some suggestions for further work in this field and some comments on the possible future of secure distributed systems.

Several annexes cover additional material including circuit diagrams, ROM firmware listings and a mathematical treatment of a general approach to primality testing and integer factorization. A glossary of terms and abbreviations is provided followed by a comprehensive bibliography and reference list. An addendum cites publications by the author relevant to this research and finally the source code of the DOS version of the prototype CSS is included on a floppy disk.

2. BACKGROUND

2.1. The Definition of and Need for Security

Generally, security refers to a complex of measures which may be broadly classified into:

(a) physical;

(b) procedural;

(c) logical;

aspects which are aimed at the

(a) prevention;

(b) detection and indication;

(c) correction;

of certain kinds of system misuse either accidental or deliberate [ECMA,1987b]. Some example considerations covered by these three aspects of security are:

Procedural Security	selecting trustworthy personnel, changing passwords regularly;
Logical Security	access controls, cryptography;
Physical Security	vaults and doorlocks, guards, screening against emanation of interpretable emissions.

2.2. Internal and External Security

Security not only addresses attacks and threats external to the system, but internal attacks from known user entities. If guarantees of *authentication* can be provided, it may be possible to devise a system in which all user entities are subject to strict access control, thus minimising the internal threat. Of course, it is virtually impossible to stop a user passing information to an attacker directly, so user trust and strong enforcement procedures are also required.

Only authorised users can obtain/provide information which will help to eliminate, as far as possible, misuse of the system, such as eavesdropping on confidential data, abuse of resources, fraudulent activity, forgery of messages, etc. The recommended range of services which a security system could provide is comprehensively addressed in [ISO,7498-2].

2.3. Secrecy and Integrity

Nearly all aspects of computer security come under two broad categories, *secrecy* and *integrity*.

> ***Secrecy*** refers to those aspects of security which prevent information *disclosure* to parties not authorised to receive it.
>
> ***Integrity*** refers to those measures which are taken to protect information from unauthorised *modification* or *destruction*.

Throughout the literature, the main emphasis is placed on secrecy, while integrity is addressed as a secondary consideration. There are two reasons for this seemingly one-sided point of view, one historical and one technical. The historical reasons arise from the fact that the vast majority of research into computer security has been funded by the United States government, whose primary concern has been the maintenance of secrecy of classified information. This tradition has persisted even in commercial applications, where classified information is not the concern and where integrity, not secrecy, is the primary goal. The technical reason for the bias is simply due to the fact that the information disclosure problem is more interesting to computer security researchers, and the literature reflects this bias [GASS,1988].

However, the tools required to protect information against modification are basically the same as (or a subset of) those required to protect it against disclosure.

The final aspect of computer security which cannot be placed in either of the two categories above is *denial of service*. This again has not traditionally been a topic of computer security research, partially for the reasons given above but also because while great strides have been made in secrecy and integrity, little progress has been made in solving the denial of service problem. This is because the problem is fundamentally much harder; preventing denial of service requires ensuring the complete functional correctness of a system - something considered by many to be unlikely to be done in the foreseeable future, if indeed it is possible at all.

2.4. <u>The System Boundary and Security Perimeter</u>

The *system* is loosely defined as the collection of components comprising the computing and communications resources over which the designers, administrators and users have some control. Everything within the system is protected by the system, and everything outside the system is unprotected. The importance is not the generic definition of the term *system*, but the definition as applicable to each case in particular. Any scheme to implement security features must clearly define the *system boundary* for the system in question, and establish a clear understanding of the threats to which the system may be exposed and against which it must defend itself.

Identification of the system boundary relies on a precise specification of the interface between the system and the outside world. The components within the system may be divided into two classes; those responsible for maintaining the security of the system, and all others. The separation of the components in this way defines an imaginary boundary called the *security perimeter*. In a dedicated secure system, such as a trusted military communications network, typically the operating system and the computer hardware will lie within the security perimeter, whereas user programs, data, terminals, modems, printers and other components which the security system protects will lie outside the security perimeter.

The characteristics and behaviour of *all* the components within the security perimeter must be carefully analysed under all foreseeable conditions, because a malfunction or deliberate functional modification of any one can lead to a security violation. Indeed, for a completely trusted system, unforeseen conditions must be accounted for as well. This may possibly be realised by ensuring that the system has a failsafe architecture whereby under fault or error conditions the system reverts to a known reference state. By contrast, the components outside the security perimeter do not require consideration, because they become subject to security constraints only at the point where they penetrate the security perimeter. In an ideal system, a malfunction within the security perimeter will have the effect of expanding the security perimeter to the system boundary, causing *all* components of the system previously outside the perimeter to become subject to the security policy within the perimeter.

In a similar manner to the definition of the system boundary, it is important to define precisely the interface across the security perimeter. This interface should be enforced by the security system. For example, the list of operating system calls which require vetting for security reasons are interfaces to the security perimeter.

2.5. Users and Trust

The *user* may be defined for security purposes as the person or application whose data and activities the system protects and whose access to information and services the system controls. Any entity who does not access the system directly, but gains indirect access through another entity is still regarded as a user.

One of the assumptions that many systems make about trust is that if a trusted user wishes to breach the trust he holds, there is little or nothing the system can do about it [GASS,1988]. Although assumptions of this sort simplify system design considerably, they also introduce considerable weakness as well. The Bell-La Padula [BELL,1973] model introduces the concept of 'no read up, no write down', as encapsulated in many military systems currently in use. Here, the user, trusted or otherwise, is prevented from disclosing information by declassification even if he wishes to do so.

2.6. Trusted Systems

Although to a certain extent human users can be trusted, the concept of trusting a computer is much more difficult to define. It is extremely difficult to write software which performs exactly as desired under all conceivable conditions, especially in large projects. System activity must therefore be supervised by a kernel of software routines which have been formally analysed in terms of security criteria. Such routines are called *trusted software*, and are typical of the routines to be found in the kernel of the security processor of a secure system.

The majority of the rest of the software on the system does not therefore require to be trusted, and this considerably simplifies the design (and hence reduces the cost) of the application software packages. In addition, items of software which may be considered to be *malign* in intent, such as *trojan horses* or *computer viruses*, may be thwarted in their attempts to penetrate the system security by close supervision of their operation and execution by the trusted software in the kernel [SPAF,1988].

2.7. Subjects, Objects and Access Control

All *activities* within a system can be considered to be sequences of operations carried on by *subjects* on *objects*. A subject is defined as any active entity capable of initiating a data manipulation within the system. At the highest conceptual level, human users are subjects, but within a system, a subject is usually considered to be a process, job, task or operation on behalf of the eventual end user. An object, therefore, is any entity manipulated by a subject in the pursuance of a task or process. Clearly, an entity can be a subject under certain conditions and an object under other conditions. For example, a computer program residing on a disk storage medium will be an object if it is moved or deleted, but will become a subject when executed and manipulating data under its own control.

It is very important that every entity within a system has a unique identifier. In the above example, distinguishing between the program as it resides on disk, and the process it becomes when executed, is important because the same program may be run simultaneously by different processes on behalf of different users, where each process possesses a unique ID. It is easy to fall into the trap of loosely identifying the program as a subject rather than as the process within which the program executes.

2.8. Distributed Secure Systems

A *secure network* is a set of communications mechanisms that provides to its subjects a specific type of service at a given protocol layer, normally the Application layer in the context of OSI. The subjects (users) are the communicating entities that use the secure network, implementing their own protocols to communicate among each other. The nature of these subject to subject protocols is of no concern to the trusted network. The network consists of all the elements that make up the protocols; the internal layers of the secure network are *transparent* to the subjects. This concept of 'hiding' functions and protocols is consistent with that of a layered protocol model.

The policy enforced by the secure network has the sole aim of determining and controlling which pairs of subjects can communicate, and which subjects can access and/or manipulate which objects. Taken together, a network of several systems - each of which contains a portion of a secure network - is a *distributed secure system*. The trusted portions of the individual systems interact via secure paths, and the untrusted portions are managed within each system in accordance with the security policy. This is akin the idea of *domains* which are discussed in Chapter 5.

When physical protection does not extend from end to end (between entities in remote systems), it is necessary to replace the physical protection with logical protection through such mechanisms as encryption. From outside the *security perimeter* the logical view of the network is the same as any other unprotected system, but the architectural view differs from the point of view that the software and hardware in the lower protocol layers need not be trusted while still maintaining security. For example, encryption in protocol layer 3 serves to provide a protected path between remote systems at layer 3, compensating for the lack of physical protection at layers 1 and 2. Protocols using encryption are also capable of providing a mechanism for *authentication*, which aims to ensure that neither of the two communicating entities are masquerading.

2.9. Mutually Suspicious Systems

Together, the systems comprising the secure distributed system constitute security *domains* which operate under a common security policy. Each domain is equally responsible for the overall security of the system. Despite the fact that systems may assure themselves of their own security within their own domain, it is often a requirement that domains communicate with each other, and possibly exchange confidential information. Each domain will therefore regard the other as lying outside its own domain and hence the scope of its own policy, and needs assurance of the trustworthiness and security 'goodness' of the other system before divulging information. Such a pair constitutes a *mutually suspicious system*.

2.10. Security Kernel on a Network

Fundamentally, the distributed systems so far discussed constitute a distributed operating system. If the secure part of that operating system in each domain is a security *kernel*, then it is necessary for the security kernels to cooperate in some way if security is to be maintained across the entire system. One way to accomplish this is to allow the kernels in the individual systems to communicate directly with each other, exchanging the necessary control information to coordinate the exchange of traffic and activities. This technique requires a trusted set of kernel to kernel protocols, and is explored fully in the Security Management Centre (SMC) concept which is central to the methodology described in this thesis. The SMC concept combines this with the technique of keeping the kernels or SMCs as autonomous as possible, thus limiting the amount of information required to be exchanged.

2.11. Trusted System Evaluation Criteria

Trusted system evaluation criteria are based mainly on the US Department of Defense *Orange Book*, after the colour of its cover [DOD,1985]. The document employs the concept of a *trusted computing base*, a combination of hardware and an operating system that supports untrusted applications and users. This concept is also central to the Comprehensive Security System described later in this thesis. The Orange Book defines seven levels of trust ranging from systems that have minimal protection features to those that provide the highest level of security. The book also attempts to define objective guidelines along which evaluation criteria can be based for both commercial and military applications.

The European Community have recently produced equivalent standards under the Harmonised Criteria of France, Germany, the Netherlands and the United Kingdom entitled Information Technology Security Evaluation Criteria (ITSEC) [ITSE,1990].

3. SECURITY IN OPEN SYSTEMS

3.1. The Concept of Open Systems

As computers have become smaller, cheaper and more numerous, there has been much interest in connecting them together to form networks and distributed systems. There is still some debate among researchers as to the exact difference in definition between a raw network and a fully distributed system [ENSL,1978], but for most purposes, the difference may be summarised as follows. A network comprises a number of system elements connected to a central processing unit (CPU) which are able to exchange data via a communications channel, whereas a distributed system takes the concept somewhat further in that it is the processing power itself which is distributed throughout the system, accessed via an operating system which is itself of a distributed nature.

Initially, connections between computers were made in a somewhat *ad hoc* way, typically with a *host* computer regarding everything else connected to it as terminals. However, the concept underlying a modern system is that due to the computing power itself being distributed throughout the network, each component is able to talk on equal terms with its peers. This distribution of processing power has the additional effect of enhancing reliability considerably, because failure of a single component will have little effect on the system as a whole.

These advantages are the two main driving forces behind the networking of computer systems. By way of example of the first, many organisations have a substantial number of computers in operation, often located far apart. A company with several factories and an administrative site may have a computer at each location to keep track of inventories, stock control, monitor productivity, do the payroll, and so on. Initially, these computers worked in isolation from one another, but management may have decided to reap the benefits of connecting them together so that information about the entire company may be extracted and correlated for management purposes. In addition, expensive peripherals such as laser printers need not be provided for each user, but be made available to all users of the network. In general, the goal of this approach is to make all programs, data, peripherals and other resources available to any user of the network without regard to physical location of either the resource or the user. Taken to its logical conclusion, this could have a very significant social impact on the whole concept of 'going to work'. It is possible that in the future many people may work entirely from home, eliminating the need for travel to and from a place of work. In addition, their choice of geographical location in which to live will have very little bearing on their ability to do a particular job. It is interesting to note that the first regional council in the United Kingdom to embrace the new ISDN public network is the Highlands and Islands Council serving some of the most remote communities in the country. In addition, the opportunities of computer-aided *open learning* are created. It may be possible to use the enhanced communication capabilities of future systems for interactive learning programs, and research effort towards this end is being carried out at the University of Plymouth using satellite technology as the communications medium.

The second goal is to provide high reliability by having alternative resources available. With unconnected computers, if the machine 'goes down' due to hardware failure for example, the local users are denied service, even though there may be substantial computing resources available elsewhere which are being under utilised at that time.

With a network, the temporary loss of a single processor is much less serious, because its users can be accommodated elsewhere until normal service is restored. For military, banking, industrial process control and other critical applications, complete loss of computing power for even a few minutes for whatever cause can be catastrophic to the business in hand.

The fundamental ability to network computers came with the change in the relative cost of computing versus communications. Until around 1970, computers were expensive compared with communication facilities. The reverse is now true. In some applications, data is generated at widely scattered points. Prior to the advent of cheap computing, it was not possible to analyse the data at each point because computers were so expensive. Raw data was sent back to a central processor for analysis. Now that the cost of a small computer is comparatively negligible, the local data can be processed, and results exchanged with other computers. The result is a computer network with the processing power distributed throughout the system.

Networks of computers are generally classified by size into Local Area networks (LANs) and Wide Area Networks (WANs). While there is no fixed boundary between the two, in general a LAN is understood to be a network serving one site such as a factory or a University campus. A WAN is a network connecting remote sites, either country-wide or even world-wide in extent.

Some researchers state that in addition to the advantages already discussed, users of networks can expect that as a result of coupling large numbers of smaller processors into large systems, simpler software designs will result [TANE,1981]. The argument for this is that in a network, it is possible to dedicate some (or all) of the processors to specialised functions, for example, database management. Instead of having one large machine multiprogrammed on a time-shared basis, each machine only performs one task at a time. By eliminating the multiprogramming, much of the software complexity associated with large mainframes is eliminated.

In summary, the concept of open systems or Open Distributed Processing (ODP) is the conceptual framework within which systems of diverse origin, application and location can interact freely if required with the attendant advantages described.

3.2. The Need for, and Attitudes Towards, Standards

The situation with many aspects of commercial engineering has traditionally been to get your own product to market first, and if anyone else wishes to produce other products compatible with yours, then they must conform to your standards or face exclusion from the marketplace. While this makes sound commercial sense from the manufacturers point of view, and indeed was the attitude and major reason for the total computer industry dominance of IBM, the situation was intolerable for users. Once a user had invested in a product from a particular company, if he ever wished to expand or enhance his system, he was compelled to make all future purchases from the same supplier. This is called *locking in* to a product range. The disadvantages are obvious. Manufacturers can go out of business, competitors can offer superior products at cheaper cost, but the user is unable to avail himself of them. In addition, users who have bought their equipment from different suppliers stand very little chance of being able to connect their systems together and so gain the advantages of networks described above.

An impressive example of the services made possible by the adoption of universally accepted standards is the international telephone system. Subscribers in different countries can make direct calls to one another as a matter of course. This is made possible by the existence of standards, agreed between the telecommunications administrations throughout the world.

Pressure from customers persuaded the various standard bodies to develop sets of common standards to which computing equipment must conform if it is to be able to claim compatibility with other similarly equipped machines. This will allow a potential customer to satisfy himself that the equipment he is considering buying will allow him to expand and enhance his system with other vendors products which also conform to the standards. Clearly, it would be in the manufacturers interests to design their equipment around these standards, or they would find few customers for their products. By far the most prevalent of these standards to emerge is the Open Systems Interconnection (OSI) framework developed under the auspices of the International Standards Organisation (ISO) based in Geneva. It is thus towards this standard that the vast majority of research effort is being directed, and the research described in this thesis is no exception.

As part of Government recognition of the central importance of information technology to a healthy and prosperous economy, a high level committee was established in 1981 to reflect the views of both suppliers of computer equipment and the wide potential user base [SIDE,1988]. This FOCUS committee is chaired by the Minister of the Department for Trade and Industry (DTI). This action acknowledged the vital need for internationally recognised and respected information technology (IT) standards, and FOCUS have the remit to advise on Government policy towards IT standards generally and to indicate what action can be effectively undertaken to encourage and assist standardisation in priority areas. FOCUS quickly established that the '...immediate and urgent priority was standardisation on OSI...', and the committee have since been consistently unremitting that there should be no let-up to the efforts applied in this area. While the Government play no part in the actual making of standards they have provided a powerful incentive to their implementation, by specifying them for their own use.

In addition to Government efforts, other powerful bodies have added their weight to the universal adoption of OSI standards. In particular, the European Computer Manufacturers Association (ECMA) have drawn up a series a proposals for Distributed Office Applications based upon the OSI framework [ECMA,1987a]. Their recommendations examine the needs of office applications of a supportive nature versus those of a productive nature, and how cooperation between these two elements can be harmonised to create a more productive and efficient office environment. ECMA have also addressed the security aspects of open systems, and their recommendations are discussed in [ECMA,1987b].

3.3. The OSI Reference Model

It is against this background of almost universal adoption of the OSI standards that the research described in this thesis is set. The fundamental reference architecture which is applicable to this research is the International Standards Organisation (ISO) Basic Reference Model [ISO,7498]. The model establishes a framework for coordinating the development of existing and future standards for the interconnection of systems. The objective of OSI is to permit the interconnection of heterogeneous computer systems so that useful communication between *application processes* may be achieved, raising the possibility that the users of *any* two computer systems may, for example:

1. exchange files;

2. exchange electronic messages (e-mail);
3. log onto each other's systems;
4. submit jobs to each other's systems.

Thus, isolated systems and 'closed' groups of systems will be opened to one another as OSI products become widely available. As a basis for the development of these standards, ISO have developed a Reference Model (ISORM), to partition the problem into discrete layers, and provide a conceptual framework for the understanding of the complex processes involved in computer communications.

3.3.1. The Seven Layer Model

The ISORM has seven layers, see Figure 3.1.:

1. application layer;
2. presentation layer;
3. session layer;
4. transport layer;
5. network layer;
6. data link layer;
7. physical layer.

The ISORM specifies the functionality of each layer, the interfaces between adjacent layers and the method of achieving layer-specific functionality between cooperating computer systems over real physical media. The communication functions of each layer are examined in turn.

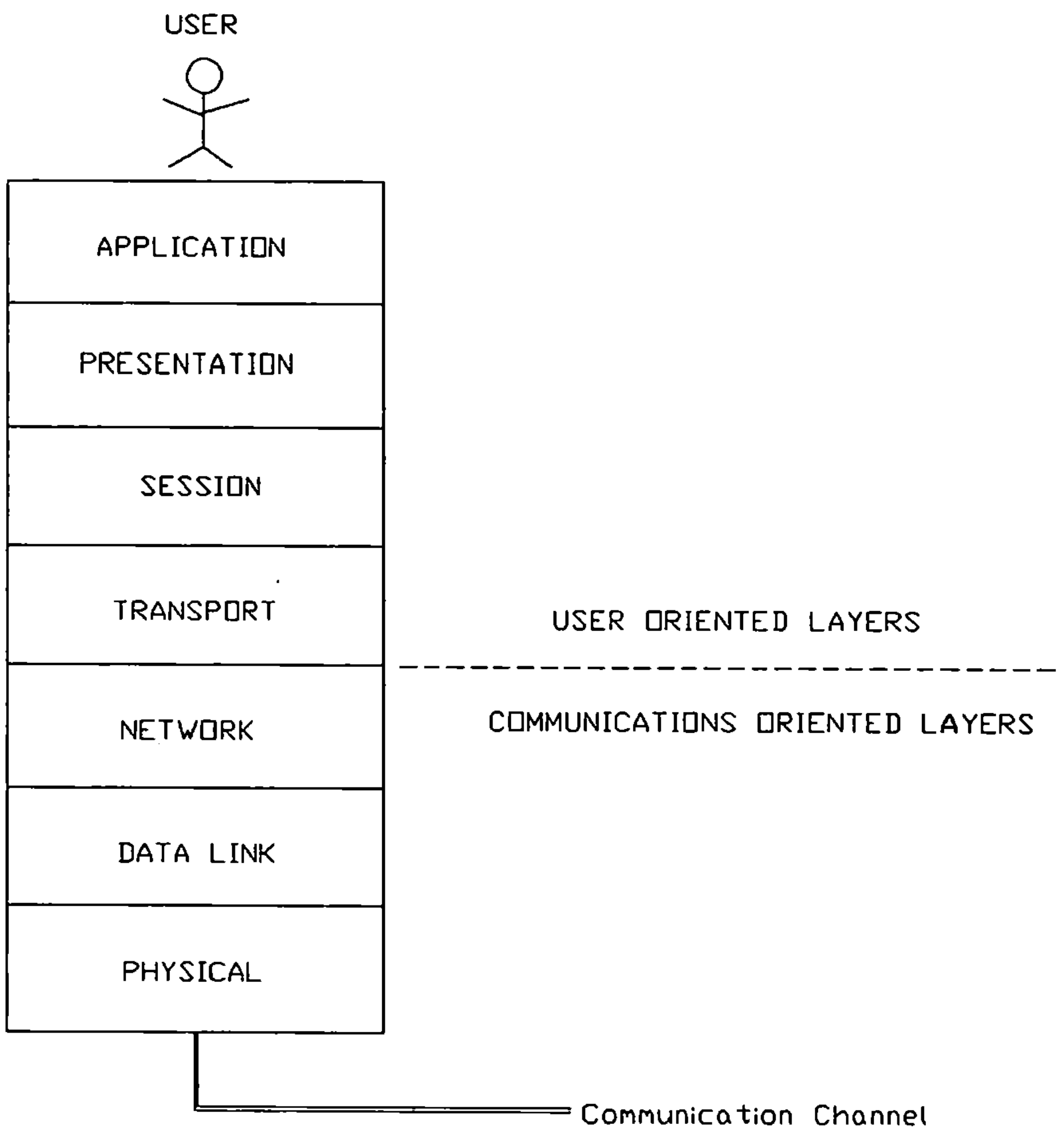


Figure 3.1. The Seven Layers of the OSIRM

APPLICATION LAYER

The goal of OSI is realized at the application layer, since it is this layer that provides the communication-based service to end users. The subordinate layers of the model exist only to support, and make possible, the activities that take place at the application layer. In this layer, all 'high-level' system-independent applications activities are performed.

These activities are controlled by entities via the local operating system which acts to interface the system-independent nature of the ISORM to the specific nature of the individual computer system. This entity is referred to as an *application agent*. An application agent may operate purely in a local, interactive mode, such as a text editor, it may provide a local user with an interface to an OSI application, or it may coexist with other application agents to provide support services to the system. It is towards this last mode of operation that the *Comprehensive Security System* to be described in Chapter 5 is directed.

As the uppermost layer of the ISORM, the application layer is unique. It differs from the lower six layers in that it alone makes OSI services available to the users of the computer system on which it resides. The layer embodies a wide range of system-independent application functions, some of which are widely applicable, well defined and standardised by OSI. These include such applications as:

1. file transfer and directory operations;
2. message handling services;
3. job transfer and remote job management.

The first of these, FTAM (File Transfer, Access and Management) is discussed in detail as an example of an OSI service in the following section. In all standardised applications, the way in which the functionality is achieved is covered by the standards. Any product, independent of specific implementation details, which conforms to the standards is assured (in theory) of the ability to interwork with all other conforming products.

Thus, the application layer is concerned with providing services, covering a wide range of applications, to the end user. All application layer activity involves the transfer of *information* over OSI between distinct cooperating computer systems. Clearly, the representation of information within the individual systems will differ in the details of data structure, syntax and so forth, but the commonality of *encoding* of the information to be transferred must be established between the cooperating systems. This is the function of the next layer.

PRESENTATION LAYER

The function of the presentation layer is to provide a common representation of application information whilst in transit between two cooperating computer systems. By way of example, consider the differences in the representation of characters across different systems. It may be that the two computer systems both use the standard ASCII (American Standard Code for Information Interchange) code for character representation. In this case, little need be done by the presentation layer other than to establish that this common coding does indeed apply. If, on the other hand, one machine uses ASCII while the other uses EBCDIC character encoding, then the presentation layer must ensure that appropriate transformations are performed on the information. To do this, it establishes by negotiation between the two computer systems, a common representational form for character information for use whilst such information is in transit between them.

In summary, while the application layer offers system-independent activity over OSI, the presentation layer exists to ensure that any information exchanged between the systems as a result of application layer activity, is in a commonly understood form. The presentation layer takes no part in the activity associated with the actual establishment and control of data communication between systems; this is provided by the next layer.

SESSION LAYER

The session layer fits between the application-oriented upper two layers, and the four lower layers which comprise the 'real-time' data communications environment. Its fundamental role is to provide services for the management and control of data flow between the two computer systems. For example, the session layer allows for the insertion of synchronisation points in the data flow of application information so the communicating processes can determine, should the flow be interrupted for some reason, the correct point at which to restart transmission, thus avoiding a wasteful rerun of the whole association. Session layer activity allows activities to be started, halted, restarted or abandoned under the (indirect) instruction of the application layer. An activity may be halted, for instance, to allow a more urgent activity to take place, and then restarted at a later time.

TRANSPORT AND LOWER LAYERS

The transport and lower three layers are not of major concern. The lowest three layers (Network layer, DataLink Layer and Physical Layer) are fully defined by the CCITT X.25 specification, an excellent tutorial summary of which can be found in [SNAR,1983]. Reliable data transmission between end systems, that is *end-to-end* communications, are assumed by the session layer.

The fundamental job of the lower layers is to deal with the problems of data transmission over a real, physical medium. In reality, this may be a landline, a public switched network, a satellite link, optical fibre and so on. Communications media differ in fundamental ways, and so must the data transmission techniques used over them. The result of the activities of the lower layers is that, although the quality of service may differ between one sub-network and another, the data is reliably transmitted and received by the communicating systems. The main job of the transport layer is to perform error handling on data transmitted across sub-networks not designed for reliable data exchange. For example, modems are used to communicate between personal computers using the public telephone system, which was designed only for voice traffic. Much of the modem's effort goes into error detection and correction for computer data which is quite alien to the telephone system. Hence, the transport layer provides the *session* with reliable data transport irrespective of the nature of the underlying sub-network.

Sub-networks using packet-switching fall into two categories, *connection oriented* and *connectionless*. Connection oriented sub-networks operate by establishing *connections* (discussed later) and exchanging data in discrete *packets*. One of more intermediate computers called *switches* may be used to direct the data packet correctly from origin to destination. Connection oriented networks generally operate at lower speeds up to around 64,000 bits per second, but can operate over almost unlimited distances. This is exemplified by the *virtual circuit* approach. They are seen as the backbone of the WANs described earlier, and are operated throughout the world by the public telegraphic systems providers as well as private concerns such as AT&T ISTEL.

By contrast, connectionless sub-networks are often capable of higher speed operation, the *datagram* approach being typical. Very high speed connectionless operation up to 10 million bits per second is sometimes possible if the nature of the physical medium which can be used for the transmission is suitable and the systems are in close proximity. This capability is associated with LAN technology, such as Ethernet.

The ISORM defines methods whereby high-level application activity can be performed between remote systems by either of the two connection modes described above.

Indeed, between some systems there may be a combination of both. A local user may access a *gateway* via the local, high-speed connectionless LAN, which gives him access to a long-haul packet-switched X.25 backbone. At the other end, a gateway onto another LAN may make the final connection to the actual receiver.

In summary, the lower four layers provide the ISORM with a means of reliable transmission of data. The physical layer is concerned with such aspects as voltage levels, pin assignments for connectors and so forth. The data link layer provides an error correcting, flow control and synchronisation framework around data for reliable transmission by the physical layer. The network layer makes use of underlying data link services to provide data transmission services across sub-networks. It is particularly concerned with *routing*, that is, establishing a route between the two computer systems, and *relaying*, the use of intermediate computer systems to provide a data flow from one sub-network to another which may be necessary as a result of the chosen route. In this way, communications facilities can be established between systems which are not even connected to the same network. The transport layer is said to operate *end-to-end* between the two computer systems, that is, without explicit involvement in any intermediate computer system that may be relaying between the sub-networks. Throughout the rest of the thesis, the term *network* is used to mean the network as seen by the transport layer, that is, a transparent, reliable communication medium across which data will flow without error or mishap.

3.3.2. Layer Cooperation

When two computer systems, or *end-systems*, are involved in an OSI communication, they conform to the standards associated with each layer of the ISORM. An implementation of a layer conforming to the standards must be capable of understanding and acting upon messages exchanged with an implementation of the same layer standard on another cooperating system. When such an implementation is *invoked* it is known as a *layer entity*. The layer is conceptualised as being split across the two end systems and represented on each by a layer entity. These two entities, called *peer* entities, will perform a layer function by exchange of messages in a coordinated manner as defined by the standard associated with that layer, see Figure 3.2.

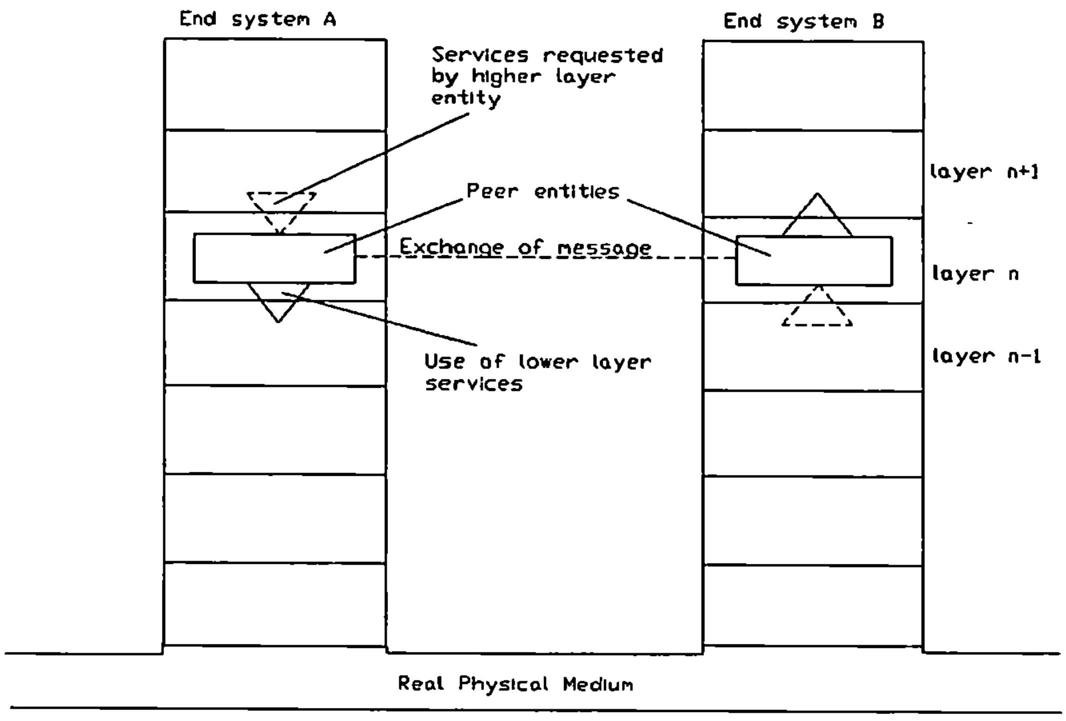


Figure 3.2. Peer Entity Association

By way of example, an application layer entity may request the reading of a file on a remote end-system. This function is achieved by an exchange of messages between peer application entities, resulting in a flow of information between application entities and hence between the file and the user. The representation of the information in common form over the communication path is the responsibility of the presentation layer, so in order to achieve the information transfer, the application layer must make use of the services of the presentation layer. The process begins with the user requesting a service of the application entity. The entity will perform the function of an exchange of messages with the peer application entity on the remote end-system in question. The initiating application entity will bring about this exchange by requesting a service from a presentation entity. The presentation entity will in turn initiate the required functions by the exchange of messages with a peer presentation entity. If this process is continued for the presentation layer's use of a session entity and so on, it will be seen that a 'cascade' is developing. This is the whole principle of communication in the OSI environment, for it is only via the physical medium that *actual* communication occurs. The communication at the higher layers between peer entities is purely a *logical* association.

3.3.3. Principles of the ISO Reference Model

As described above, all seven layers of the ISORM must be involved in any OSI application activity. Entities of each layer must be present at each cooperating end system to play their role in the mechanism of cooperation. The terminology used when referring to the architectural principles of the ISORM is now discussed. A layer n user on end system A requests of a layer n entity an n-specific service. Note that only if the layer happened to be the application layer would the user be a 'real' user! In other instances, the 'user' would be a layer $n+1$ entity. The realisation of this service over OSI requires the cooperation of two peer n entities, one on each of the cooperating end-to-end systems, and involves the exchange of a control message between the peer n entities. This message indicates the nature of the required cooperation and carries with it any parameters associated with the particular service, encoded in a precisely defined format. Direct message exchange therefore is only *notional*, since the only physical route between end systems is provided via the physical medium, and access to this medium is only via the physical layer. The *n* entity can only achieve peer entity association by calling on the next lower *n-1* layer, to act on its behalf. In general, an *n* entity operates by utilising services on an *n-1* entity.

The process is now repeated for the *n-1* entity. Having been invoked by the *n* entity, the *n-1* entity satisfies the service request by cooperating with its peer on the remote end system. To do this it constructs an *n-1* control message. The *n* control message is passed to the *n-1* entity as part of the service request and is packaged into the *n-1* control message. If *n-1* is not the layer at which physical data exchange between the end systems can occur, the procedure is repeated with a service request by the *n-1* entity acting as a user of an *n-2* entity. This continues until the physical layer is reached, at which data exchange consists of a series of *encapsulated* control messages.

On reaching the receiver, the process is reversed. The control message associated with each layer is examined and the activity implicit in that message is performed. It may be that the control message may just instruct the entity to pass on the residual part of the message, which contains encapsulated control messages for higher layers, without any other action. This case is known as *normal data transfer*. Other types of control message instruct an entity to perform some layer specific activity. A point of importance is that each entity issues a report to its user (the layer above) which corresponds to the original request issued by the peer user. This report, called an *indication*, implies that the requested layer service has been performed by the cooperating peer entities. Together with the indication the layer passes the residual part of the control message for action by the higher layer.

3.3.4. OSI Protocols and Primitives

To summarise, a layer comprises a set of functions which, when activated by user requests, provides the realisation of the services. Every function has its own associated control message which conveys instructions between cooperating peer entities in order that the required function can be performed by the peer entity without ambiguity. Every layer is defined by a precise set of functions and associated control messages. These functions, the format and parameters of the control messages, and the actions to be taken on receipt of a control message or service request, are defined as the layer *protocol specification*. A layer user communicates a request for a service to a layer entity, or receives an indication of a service invoked by a remote peer user, in a *service primitive*. Every service element has a set of service primitives which are defined in the service definition. The user, known as a *service user* (S user), requests a service by means of a service primitive. As well as this request, the S user may include the control message or *protocol data unit* (PDU) that it wishes to convey to its peer. This results in an *n* entity, which as an *n-1* S user, issuing an *n-1* service primitive to request a service of an *n-1* entity. The *n-1* S user may include any *n* PDU that it wishes to convey to its peer. This 'user data', passed between the *n* entity and the *n-1* entity as part of a service primitive call, is known as a *service data unit* (SDU). In this case, the *n* PDU is an *n-1* SDU. This situation is summarised in Figure 3.3.

Again, at the receiving end, the opposite is performed. The *n-2* entity strips off the protocol control information and acts according to the instructions implicit in it. It then issues an *n-2* service primitive to indicate to the *n-2* S user (an *n-1* entity) that activity has been performed as a result of an *n-2* S user request on the sending system. The *n-2* SDU which contains the residue of the *n-2* PDU (ie the *n-1* PDU), accompanies this indication. The *n-1* entity acts in precisely the same manner resulting in an *n* entity

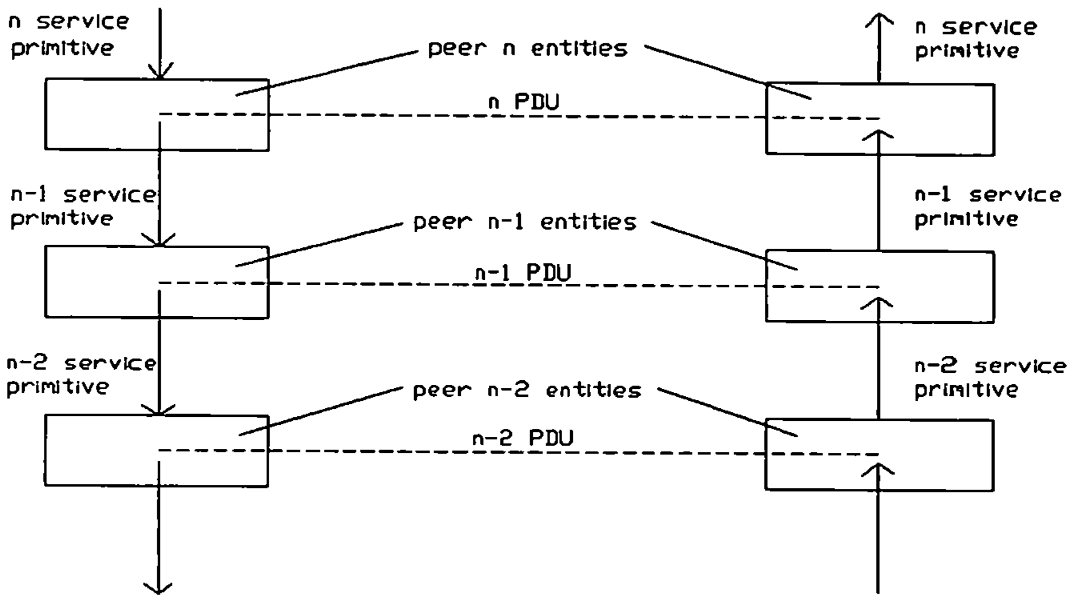


Figure 3.3. Peer cooperation via subordinate layer entities

receiving the *n* PDU (in an *n-1* SDU) which the initiating *n* entity sought to convey. This continues upwards until the *n* entity issues an *n* service primitive to the *n* S user, indicating that the remotely initiated activity has occurred. Thus, the objective of the original *n* S user request, issued on the initiating system, has been fulfilled. The service primitives that are used to request services or indicate the occurrence of services are called a *request* service primitive and an *indication* service primitive respectively.

For the above sequence of events to occur without problem, it is vitally important that the interfaces to each layer entity be precisely defined. The entry point to each is defined via a *service access point* (SAP). An *n* entity offers services to an *n+1* entity through an *n* SAP.

Thus, the consequence of issuing a request service primitive on an end system is the activation of a service element, resulting in the exchange of a PDU with a peer layer entity. This brings about a cooperative activity and results in a *indication service* *primitive* on the remote end system. However, the initiating user has no explicit confirmation that the service has been performed. A service of this nature is therefore called an *unconfirmed* service. There are many instances where confirmation is desirable or even necessary to convey some information resulting from the activity. A service of this type is called a *confirmed* service, and associated with the process of confirmation are two further service primitives. These are the *response* and *confirm* service primitives. The use of these primitives is indicated in Figure 3.4.

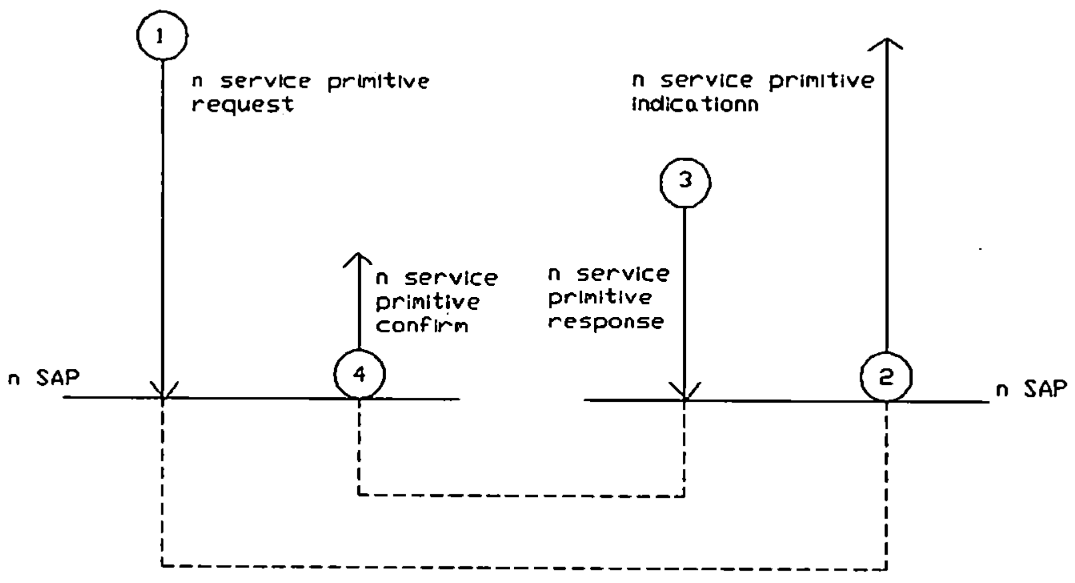

1
n service primitive request
n service primitive indicationn
3
n service primitive confirm
n service primitive response
n SAP
4
2
n SAP

Figure 3.4. Confirmed Service Primitive Usage

A service element can therefore have a maximum of four service primitives associated with it. If it provides a confirmed service there will be four, if unconfirmed, two. Certain service elements have a single associated service primitive, called an *indication*. This may occur in a situation where a communications failure occurs in a lower layer, that is, in the *n* S provider. The *n* S provider detects the failure and issues an *n* indication service primitive to both *n* S users.

3.3.5. Connection Oriented Operation

In connection oriented operation, the first task of the session S (SS) user, triggered by some application request, is to establish a data communications path between itself and the peer SS on the remote end user. This is known as *session connection*. It is a relationship between peer SS users, which once established and until released, constitutes a channel through which SSDUs can flow. The session connection can be released as a result of an SS user request, or as a result of a (possibly catastrophic) event in a lower layer. In connection oriented operation there is no way for SSDUs to be exchanged between SS users other than over an established session connection. By way of analogy, many readers will be familiar with the idea of logging onto a remote mainframe from a terminal. At the 'PAD' prompt the command CALL <HOSTNAME> is issued, and a name and password have to be entered before the session is established. This will continue until the user logs off and the session is terminated. We now examine the steps involved in a session connection establishment.

Using the terminology introduced in the previous sections, a session is established as follows. On receipt of a session CONNECT-REQUEST from the SS user, the session service element concerned with establishing the connection constructs a CONNECT SPDU. It must then utilise a transport layer service to bring about the transfer of the SPDU to its peer on the remote end system. At this point, the transport entity on the initiating end system is 'idle', that is, there is no transport connection established between peer transport service (TS) users. As discussed, in a connection oriented environment, *n* SDUs cannot flow between S users until connection has been established, and so we encounter a problem in that the session entity cannot use the transport service to transfer the CONNECT SPDU (as a TSDU) until a transport connection is established. The session entity must retain the CONNECT SPDU in the meantime, and issue a transport connect request service primitive to the transport entity. The session entity then leaves the 'idle' state and goes to the 'connecting' state. In this state, it cannot accept any SSDUs for transfer.

This process is repeated in the transport layer, with the transport entity holding on to its CONNECT TPDU and entering the 'connecting' state until such time as a network connection is established between it and its peer. When the network entity receives a NPDU and issues a network CONNECT-INDICATION service primitive to the transport entity, the receiving transport entity must decide whether to accept or reject the proposed connection between itself and the initiating transport entity. Security constraints or resource problems may dictate rejection. If, however, it is accepted, the receiving network entity builds a CONNECTION-ACCEPT NPDU which it transmits to the initiating peer, which informs the transport entity of this fact in a network CONFIRM service primitive. The initiating transport entity can now 'release' the held CONNECT-TPDU which it conveys to its peer. A similar sequence ensues in the higher layers, the session entity receiving a transport CONNECT-CONFIRM. Again, this is followed by the session release of the 'held' SPDU which finally causes the session to be established and the initiating SS user informed by a session CONNECT-CONFIRM. At this point, the SS user knows that a data communication channel has been established between it and its peer, and SSDU exchange and associated session services are now available. The activity to be undertaken by the application entity can therefore begin.

3.3.6. Connectionless Operation

A connectionless mode of operation is one in which an SPDU transfer service can be requested of an entity at any time. There is no requirement for a connection between

SS users to be established before session services can be utilised. Thus, connectionless entities are never strictly 'idle' since they are always 'ready'. Since no connection exists between SS users, the route (defined by the network address together with the SAP addresses) must be included explicitly with every SSDU transfer request. Clearly, a connection oriented *n* entity cannot cooperate with a connectionless *n* entity. By contrast, however, a connection oriented service can be offered over a connectionless network service.

3.3.7. Structure of an Application Entity

We now consider the structure of an OSI application entity in more detail. Within the OSI environment, an application *process* is represented by one or more application entities. That is, each application entity represents a different aspect of the communication behaviour of the application process. Each application entity has to request an *application service element* (ASE) to perform tasks for it. A *service element* is a *primitive* defined at the interface between two adjacent layers. An ASE is a set of functionalities that supports a typical application. This concept is represented in Figure 3.5. Each ASE stands for one implementation of an OSI service on the system. For example, ASE1 could be X.400 electronic mail, and ASE2 could be FTAM.

At any given instant, the application process works with either ASE1 or ASE2. The single controlling function routes the application process to the appropriate ASE. In the case of several OSI communications (a *multiple session*), the multiple association control function manages coordination of the application entities. Each application entity contains one *user* (element) and a set of application service elements. The specific combination of these different elements determines the type of application entity.

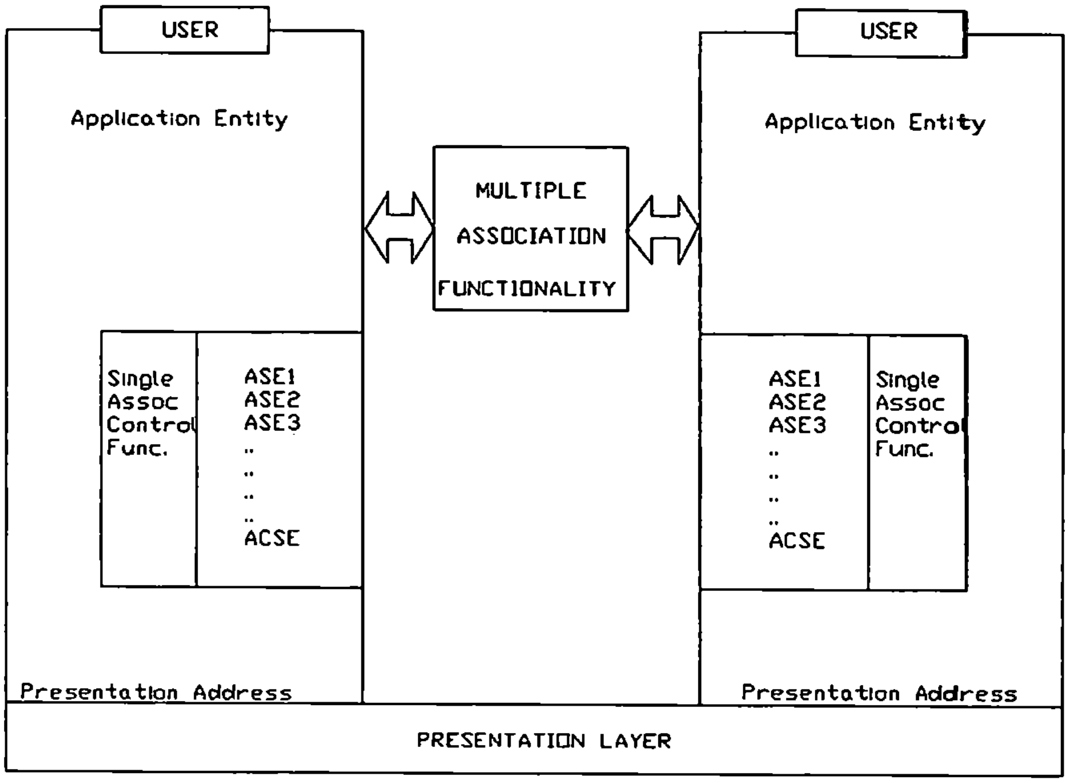


Figure 3.5. Use of Application Service Entities

The user element represents, or acts on behalf of, the application processes and allows the ASEs to communicate with other application entities. In general, the user element represents the ultimate source and destination of all information transfer. An ASE is a coherent set of integrated functions that allows application entities to interoperate for a specific purpose. ASEs may be used independently or in combination to meet specific information processing goals. The X.400 and FTAM ASEs are already defined, and FTAM is discussed in more detail in the next section. One particular type of ASE, the *association control service element* (ACSE) facilitates other ASEs working together.

As of the very latest release of standards [ISO,7498] only the term ASE is in use; the terms CASE and SASE are no longer in frequent currency. A CASE, or *common* ASE, represented the common functions needed for different jobs, and a SASE, or *specific*

ASE was used to differentiate FTAM from MHS and other kinds of defined application protocols. (It must be pointed out that, to date, no satisfactory objective criteria have been established for distinguishing a CASE from a SASE).

Entities in an OSI environment must be addressable, and moreover, an application entity may be able to support one or more associations. To accomplish this, each application entity is attached to a *presentation address*, which points to one or more *presentation service access points* (PSAPs). At any time, the application is bound to the presentation address of the PSAP to which the application entity is attached.

3.4. FTAM - A Typical OSI Application

Having established the basic framework of the OSIRM, the important FTAM (File Transfer, Access and Management) service is discussed as an example of a typical OSI application.

The ISO FTAM standard is defined in [ISO,8571]. The standard introduces the concept of the *virtual filestore* which is central to remote file manipulation. FTAM offers three modes of file manipulation: *transfer*, *access* and *management*. File transfer is the movement, over OSI, of a complete file between two filestores on different end systems. File access is the reading, writing or deleting of parts of files residing on a remote end system. File management involves aspects of file handling such as amendment of attributes on files in a remote filestore. Peer application FTAM entities provide the services to make these services possible. Clearly, there is a master and slave relationship between the cooperating FTAM entities. The master, or initiating FTAM process is invoked by a local user request, and supplied with information about the remote files to be operated upon. The slave, or responding FTAM process is awakened on the remote end system to become the cooperating peer; its activity is governed entirely by the requirements of the initiating FTAM process.

If it is required to work with files on the *local* filestore, it can be safely assumed that the local FTAM entity has knowledge of the structure of the store. Since OSI is designed as a system-independent concept, the FTAM entity is unlikely to have any knowledge of the structure of a remote filestore. Its only contact with such a store will be a *protocol data unit* (PDU) exchange conveyed over OSI between itself and its peer entity. Therefore, any reference to the remote filestore cannot be in implementation specific terms, but only in terms of some *generalised* filestore which is applicable by mapping to any specific filestore. This generalised filestore is known as a *virtual filestore* and models all possible filestores. FTAM SASEs relate to the virtual filestore and *not* to any specific instances of a real filestore.

The virtual filestore is a representation of a real filestore within the OSI environment. Within the virtual store, a file is represented by the following elements:

1. a *unique* filename, allowing identification without ambiguity;
2. attributes, expressing such properties as accounting information and history, such as time and date of creation;
3. attributes defining the logical structure of the information stored within the file.

The elements described above remain with the file throughout its life unless specifically modified. In addition, however, there are a number of *activity attributes* which are transitory, and reflect the current status of the file within the context of FTAM. These are:

current access request;

current initiator identity;

current access passwords;

current calling application entity title;

current account;

current responding entity title;

current access context;

current concurrency control;

current location;

current processing mode.

FTAM offers a file service which is provided by a large number of SASEs. An association between peer FTAM processes is achieved by the use of an ACSE and is generally referred to as an *FTAM association*. The service elements associated with FTAM can be grouped into the following areas:

kernel;

read;

write;

file access;

limited management;

enhanced management;

grouping;

recovery;

restart.

For example, kernel and write service elements together constitute a *regime* that permits the user on the initiating end system to transfer a complete file to the responding filestore. Because of the unusually large number of elements in FTAM, the possibility of disagreement between cooperating peer processes is increased and so to simplify negotiation, the standard defines a set of *service classes*. Each class defines a set of service elements designed to address a typical application requirement. In each class, the presence of certain elements is mandatory, while other are optional. The following service classes are defined:

1. **TRANSFER** allows movement of files or parts of files between end systems;
2. **MANAGEMENT** allows reading and modification of attributes;
3. **TRANSFER & MANAGEMENT** combines the first two classes;
4. **ACCESS** permits the location of a specific part of a file, which can then be read, written or erased;
5. **TRANSFER & ACCESS** combines classes (1) and (4);
6. **UNCONSTRAINED** leaves the selection of service elements open to the negotiating peer entities.

An FTAM association regime is established between peer FTAM processes by use of the F-INITIALISE service element, and terminated by the use of either F-TERMINATE or F-ABORT. The sequences of service elements possible for various FTAM activities are given in diagrammatic form throughout the discussion. In these diagrams, time flows from left to right, and the use of a service element is indicated by a vertical line. For example, an FTAM association regime is represented by Figure 3.6.

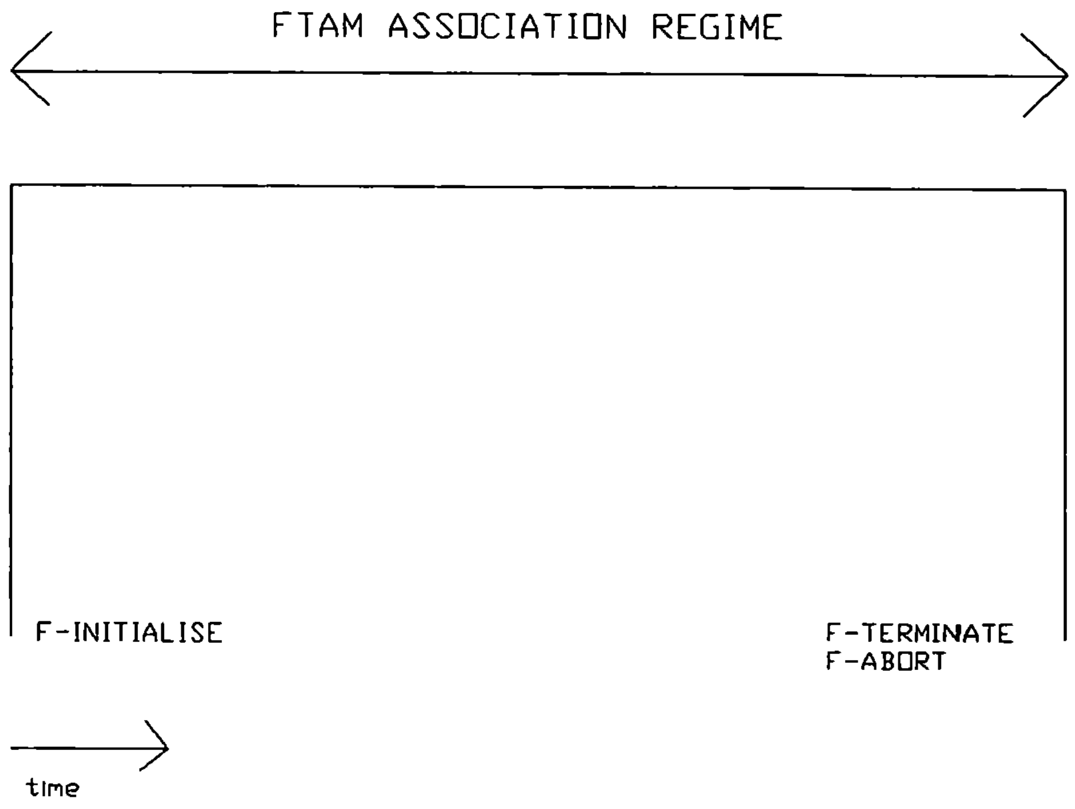


Figure 3.6. FTAM Association Regime

In this example, F-INITIALISE offers a confirmed service bringing about, by negotiation, an association regime. F-TERMINATE brings about an orderly closing of the association regime, but is only available to the originator of the association. It has only a single parameter used by the responding end system to convey any housekeeping details, such as costs, to the initiator. Either party to the association, however, can bring about the 'panic' closing of the association (which is far less orderly) by using F-ABORT.

Consider Figure 3.7. overleaf. Filestore management is a generic term for activity applied by the initiator to the filestore at the responder. To be able to make a particular file the object of subsequent activity, it must first be selected (if it already exists) or created (if it does not already exist). The selected file is assumed to be the object of all subsequent activity until the regime is closed. This, then, constitutes the first of the *inner regimes*.

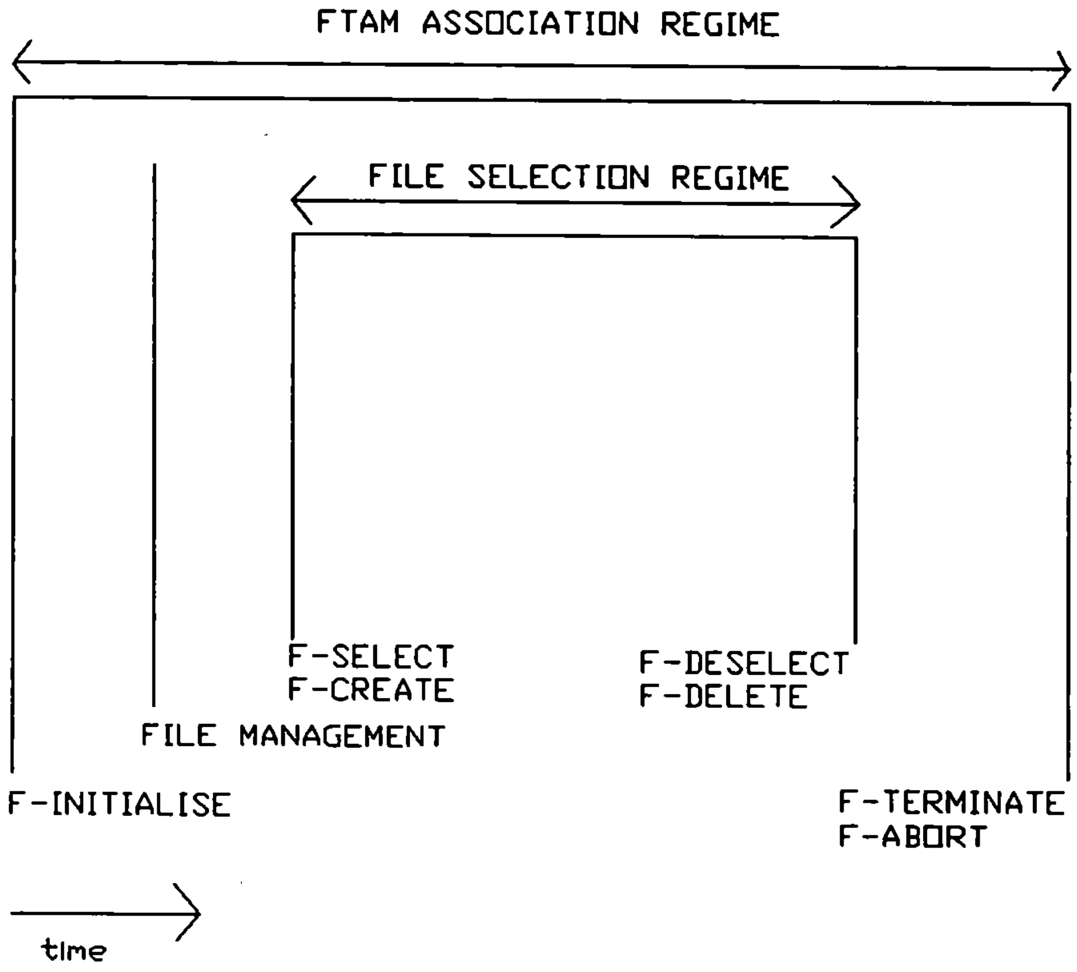


Figure 3.7. Within the FTAM Association Regime

Within this regime, it may be desired to open a file for various purposes. The file open regime is depicted in Figure 3.8.

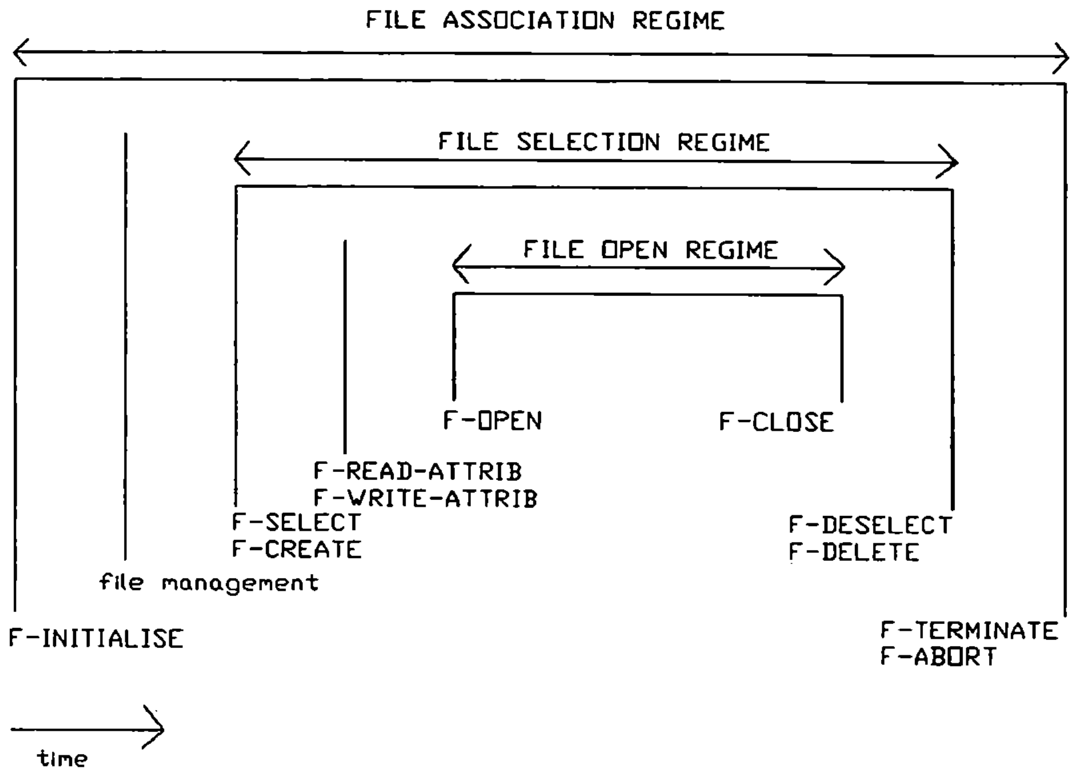


Figure 3.8. File Open Regime

It can be seen from the diagram that activities such as reading and writing file attributes can be performed within the file selection regime while no further inner regime is in force. This is in keeping with familiar operating systems such as DOS, where the getting and setting of file attributes *must* be performed on a file which has been *assigned* but not *opened*. Before the *contents* of any file can be manipulated, however, the file must be opened. In FTAM, it is opened by use of F-OPEN, and closed when finished with by use of F-CLOSE, both of which are *confirmed* services. (This is essential, as otherwise the initiating process could waste resources writing to a file it assumed has been opened correctly, but in fact has failed to open for some reason).

Once the selection regime has been established, and file activity requiring the transfer of data units to or from the file is required, it is essential that the structure of the file be understood by both initiator and responder. Such understanding is in terms of the virtual filestore described earlier. Consider the case where the initiator has used F-CREATE to create a new file in the respondent's filestore. At this stage, the responder will not know the intended structure of the file. Similarly, when the initiator wishes to read data units from the responder's file, he will not at the outset have any knowledge of the structure of that file other than the information being kept with the file in the respondent's filestore. One of the main purposes, therefore, of F-OPEN is to convey the structural information between peer FTAM processes in the appropriate direction, initiator to responder or vice versa. This is done by use of the F-OPEN **contents type** parameter.

Finally, as far as FTAM is concerned, the data unit (DU) is the smallest unit of data that can be transmitted across OSI. This may not correspond to the record size of elements in the file, which may be unstructured (as in the case of text files), a sequential flat file (as in the case of an untyped file), or an ordered flat file (as with files of records). It is the responsibility of the responding FTAM entity to map the DU onto the record size of the file in his filestore. The activity within the file open regime can now be considered, and is shown in Figure 3.9. The SASE's concerned with activity in this regime can be divided into two groups, those associated with management services to the initiator, and those concerned with the transfer of data between end system filestores. Examples of the management services are F-LOCATE and F-ERASE. Note in passing that this is in sharp contrast to operating systems such as DOS, where *erase* must never be used on an open file. Examples of the transfer services include F-READ and F-WRITE. The regime is terminated by use of F-TRANSFER-END. This does not close the file, but simply delimits the transfer of data to that point; any number of sequential read/writes are possible.

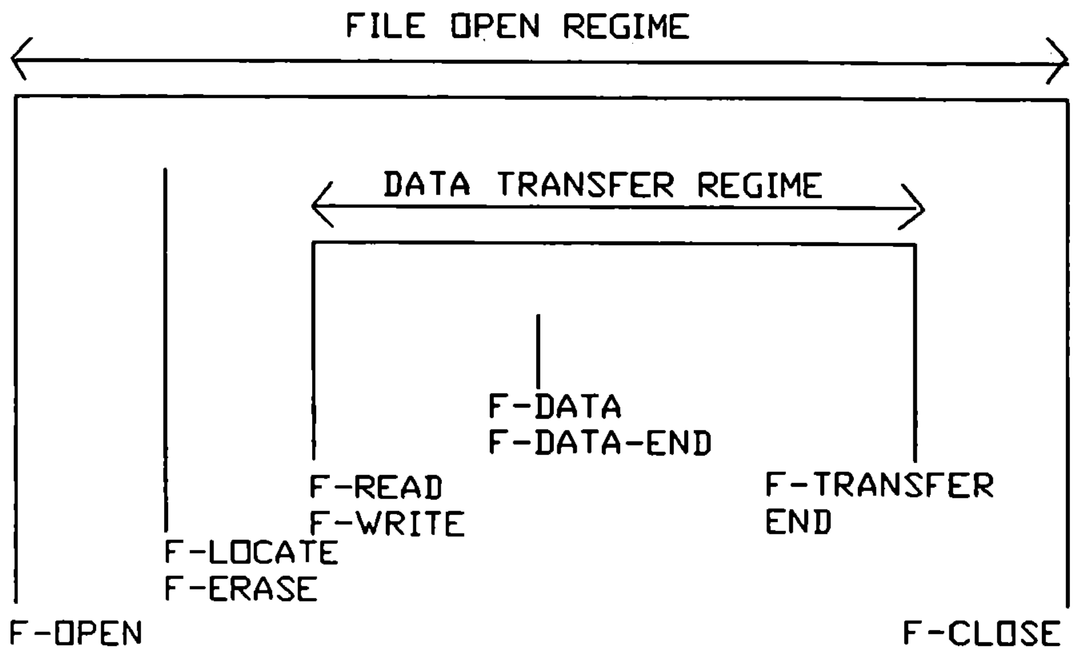


Figure 3.9. Within the File Open Regime

Due to its unique and complex nature, the convention adopted for the naming of FTAM PDUs (FPDUs) is slightly different from the other services. Here, the FPDU associated with an activity is named after the associated service layer primitives. For example, the FPDUs associated with the F-OPEN service element are F-OPEN REQUEST FPDU and F-OPEN RESPONSE FPDU.

3.5. <u>The OSI Security Architecture</u>

At various times, security controls must be established in order to protect the information exchanged between the application processes. Such controls should make the cost of obtaining or modifying the data greater than the potential value of doing so, or make the time required to obtain the data so great that the value of the data is lost.

That part of the Basic Reference Model which is concerned with the provision of security services is called the OSI Security Architecture [ISO,7498-2]. This architecture defines the general security-related architectural elements which can be applied appropriately in the circumstances for which protection of communication between open systems is required. It establishes, within the framework of the Reference Model, guidelines and constraints to improve existing standards or to develop new standards in the context of OSI in order to allow secure communications and thus provide a consistent approach to security in OSI.

Basic security *services* and *mechanisms* and their appropriate placement in the OSI seven-layer model have been identified. In addition, the architectural relationships of the security services and mechanisms to the Basic Reference Model have been identified. The standard admits that additional security measures may be needed in end systems in order to meet specific requirements. The fundamental architecture of the Comprehensive Security System to be described builds on the ISO references, and extends them to point where a useful, practical system may be implemented.

3.6. Placement of Security Provisions within OSI

The ISORM Security Architecture [ISO,7498-2] makes recommendations concerning the possible placement of security mechanisms within the framework of the Reference Model [ISO,7498], and identifies the architectural relationships of the security mechanisms to the Basic Reference Model. The standard also points out, however, that additional security measures may be needed for various application contexts, but defines these as being outside the scope of the standards. This *extended* security architecture is discussed more fully in the next section.

The layers at which the various security facilities proposed by the standard could be implemented are summarised in Figure 3.10. which is reproduced from the ISO standard.

Figure 3.10. Recommended Placement of Security Mechanisms within the Seven Layer Model

				Layer			
Service	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Peer Entity Authentication	□	□	Y	Y	□	□	Y
Data Origin Authentication	□	□	Y	Y	□	□	Y
Access Control	□	□	Y	Y	□	□	Y
Connection Confidentiality	Y	Y	Y	Y	□	□	Y
Connectionless Confidentiality	□	Y	Y	Y	□	□	Y
Traffic Flow Confidentiality	Y	□	Y	□	□	□	Y
Connection Integrity	□	□	□	Y	□	□	Y
Connectionless Integrity	□	□	Y	Y	□	□	Y
Non Repudiation of Origin	□	□	□	□	□	□	Y
Non Repudiation of Delivery	□	□	□	□	□	□	Y

As is clear from the figure, although many of the security facilities can be provided at a number of different layers, nearly every facility can be offered at the application layer. The only major exception is traffic analysis. This observation is central to the concept of the research to be described in later chapters.

The Security Architecture also defines some of the security management concepts used in this research. In particular, the *Security Management Information Base* (SMIB) is defined as the conceptual repository for all security-relevant information needed by open systems. It in no way constrains the implementation of the storage, nor does it imply that it resides in one physical location. This allows the concept of a *distributed SMIB* to be introduced.

3.7. <u>Requirement for Extensions to the OSI Security Architecture</u>

Although the security architecture described in the standards permits the placement of all the proposed security facilities at the application layer, it says nothing about how this may be achieved. In practice, it is necessary to extend significantly the concepts outlined in the standard to permit the successful development of a security system that will admit all the possibilities mentioned. This has been designated the *Extended ISO Security Architecture* and has been described by the author and others in [MUFT,1992].

3.8. <u>Current Approaches to Security in Open Systems</u>

Currently, if a particular application requires security services, these are generally constructed by hardware/software means into the application itself from conception. In a system where there are several normal (insecure) applications, and one or two secure services, this approach is quite satisfactory.

By contrast, in a system where there are many possible applications, as with ODP, a large number of which may require security facilities, it is clearly wasteful for each application to provide a complete set of security services for its own private use, when a majority subset of the services could be common to most, if not all, applications.

A general system was considered early in the research in an attempt to overcome this problem of duplicated services, by seeking to intercept all input and output (both data and control) to and from applications, and to impose security functions upon the application by redirection of the data via a security system kernel. In principle, all software application packages should be written to standardised specifications. For example, Application Layer entities within the ISO OSI seven-layer model should conform to the ISO reference model [ISO,7498]. On investigation, however, this is far from the case in practice. Had all existing software adhered strictly to specifications, it may have been possible (albeit very tedious) to implement such an I/O redirection system, which could cope with the widely differing interface, data and control requirements of all the various applications. Since the majority of software applications are written as an amorphous entity, with no obvious interface standards, the concept was discarded as impractical. Even within the context of a local area network of Personal Computers, running MS DOS for example, the amount of operating system interrupt handling to account for all DOS file I/O alone, proves to be an extremely difficult task.

The concept of providing a security system which is independent of, but available to, specific applications on request, is therefore only possible if the applications themselves are modified to include a standardised *application program interface* (API). The requests for security services, control and data information and any other data must flow in a rigidly defined manner across the interface which shall enable analysis of the data flow protocols for formal demonstration of the strengths and weaknesses of the system to be made.

4. FORMAL ANALYSIS AND VERIFICATION

4.1. Requirement to Evaluate Security

The requirement to evaluate formally the security of a system arises for two main reasons. Firstly, if it is possible to obtain a measure of the 'figure of merit' or 'goodness' of the system, it may be possible to estimate its overall effectiveness for the task it is required to perform, and help identify weaknesses. Once identified, shortcomings in the system, either in policy or implementation, can perhaps be overcome. Secondly, it is desirable that different systems can be compared and contrasted, and has the valuable commercial effect of allowing a potential customer of a system to evaluate rival products against each other. Once having made a final decision, it then gives the user a measure of confidence in the system he has bought.

In contrast to the refinement of cryptographic algorithms, very little orderly development has taken place in the study of computer system security models. Up to 1983, the comprehensive survey by DeMilo *et al* [AMS,1983] expressed the view that it was certain that the central issues had not yet been clearly articulated. Since then, a considerable effort has been made in a number of areas, but the majority of research is aimed at specific problem areas such as authentication, access control and so forth. Very little appears in the current literature concerning the provision of a comprehensive set of security services as an integrated whole.

4.2. Security Models

4.2.1. Role of a Security Model

Success in achieving a high level of security in a system is fundamentally dependent on the degree of care in the design and implementation. Before anything, it is essential to understand clearly the requirement of what is to be achieved, and the purpose of the *security model* is to express those requirements *precisely*. The model, therefore, should ideally have the following properties [GASS,1988], although it is difficult to simultaneously achieve all of them:

1. be precise and unambiguous;
2. be abstract;
3. be generic;
4. be a true and accurate representation of security policy.

The reasons for requirements (1) and (4) are self evident. The reason for requirement (2) is that it is necessary only to *model* the security *properties* of a system, and not the functions. It is important to avoid the tendency to confuse the model with the formal specification by including too many functional properties of the system that are irrelevant to security policy. The reason for requirement (3) is because the model must deal only with security properties, and does not unduly constrain the functions of the system or its implementation.

For high security systems, such as those required by the military and financial sectors, the system is often based on a security *kernel*, and the requirement for precision is satisfied by writing the model in a formal mathematical notation. Although the concept of modelling does not always require the use of mathematical techniques, these are generally the most powerful tools available, especially if the analysis is to be applied to a number of quite disparate systems with a view to comparison of models.

Security systems in current use have been developed in one of two ways. For low grade, *ad hoc* security arrangements, the general route has been *informal* [GASS,1988]. This methodology generally involves writing code to produce an implementation of the perceived requirement first, followed by testing to see if it meets the requirements as specified. Finally, the security requirements are shown to exist through demonstration. The methodology used (and indeed insisted upon) for high security systems is the *formal* route. This involves the production of an abstract model, which is then formally *proved* to map to a formal specification. The formal specification is then in turn proved to map onto the actual implementation. By maintaining an unbroken chain of validation techniques at every stage, the implementation is shown to fulfil the required security policy of the original model. This significantly raises the assurance level of the system. Nonetheless, the gap between a formal specification and the actual implementation (or implementations across a number of hardware and software platforms) is immense, and this gap is sometimes filled by less formal justification methods. This is a point of weakness, and must be carefully addressed to minimise the possibility of error.

4.2.2. Practical Applications of a Model

One of the criticisms levelled at formal modelling is the highly abstract nature of the process. It is possible to carry the modelling details and mathematical formalism to the point where the model does not help in the design of the system. Generally, when the functional specification is written, not enough is known to specify every aspect of the system's behaviour at that point. Indeed, an overly detailed functional specification at the early stage can unnecessarily constrain the design.

There are certain aspects, however, which cannot be left to chance or the judgement of the implementors. Subtle security flaws such as *covert channels* can be overlooked. The function of the security model is to assist in the writing of the functional specification. With care, it is possible to constrain the system to the point where the necessary precautions have been specified, without constraining the functions.

In summary, the *functional specification* serves as a guide to the *functions* of the system, whereas the *model* serves as a guide to the *security-relevant behaviour* of the functions.

4.2.3. Types of Security Model

It is difficult to classify security models, because each is very different from every other. Only comparatively few have achieved wide publicity, and even fewer have actually been used as the basis for real implementations [LAND,1981] and [MILL,1984].

A *state-machine* model describes a system as an abstract mathematical state machine. In such a model, *state variables* represent the state of the machine at any given instant, and *transition functions* describe how the variables may be changed. This basic idea is quite old, and has firm theoretical basis in the work of Turing and others [TURI,1936]. Hitherto, it has not been applied to large systems because modelling all the possible state variables of a large system is infeasible. The new reduction technique described in Chapter 6, however, has made possible the analysis of large systems by this methodology for the first time, and is the basis for the formal verification technique applied to the CSS.

The *access matrix model* [HARR,1976] is a state machine model that represents the security state of the system at any instant by a matrix array containing one row per system subject, and one column per system object. The entry at each intersection specifies the allowed access rights of any subject to any object.

A variant on the access matrix model is the *information flow model* [DENN,1983], which rather than checking the rights of subjects to access objects, seeks to control the flow of information from one object into another object, constrained according to the two objects' security attributes.

Another type of model is the *non-interference model* where subjects in one *domain* are prevented from affecting one another in a way which violates security policy [GOGU,1982]. This model is currently undergoing development by Honeywell in the Secure ADA Target research project [BOEB,1985].

Clark and Wilson [CLAR,1987] have addressed the *integrity* needs of commercial data processing and highlighted the differences between these requirements and the more traditional military concern with secrecy.

Varadharajan has focussed on the problem of *authentication* using state-machine techniques [VARA,1988].

4.3. State-Machine Models

The type of model which is of interest to the development of the Comprehensive Security System (described in Chapter 5) is the *state-machine* model, and we consider this in more detail.

4.3.1. Development of a State-Machine Security Model

The development of a state machine model comprises a number of stages:

(1) definition of the security-relevant *state variables*;

(2) definition of the conditions for a secure state. Mathematically, this definition is an *invariant* that expresses relationships between values of state variables that MUST be maintained during *state transitions*;

(3) definition of the state *transition functions*;

(4) proof that the functions maintain a secure state, eg. if the system is in a secure state before an operation, it will remain secure after the operation;

(5) definition of the *initial state*;

(6) proof that the initial state is secure in terms of stage (2).

4.3.2. An Example

A requirement from security policy might be an implementation of the Bell-La Padula military model [BELL,1973] of *no read up & no write down* restrictions.

POLICY (a): "A person may read a document only if the person's clearance is greater than or equal to the classification of the document".

POLICY (b): "A person may only write a document if the classification of the document is greater than or equal to the clearance of the person".

Before the state machine development steps can be followed, it is necessary to develop computer *abstractions* of the *elements* of the security policy, and then restate the policy in terms of those abstractions.

Real-World	***Abstraction***
person	subject
document	object
clearance	capability
classification	token

Restating the policy in terms of *properties*

Property (a): "A subject may only access an object in read mode *iff* the capability of the subject is greater than or equal to the token of the object".

Property (b): "A subject may only access an object in write mode *iff* the token of the object is greater than or equal to the capability of the subject".

Step 1 : *Define the state variables*

S	= set of current subjects s
O	= set of current objects o
acl(s)	= capability of subject s
tok(o)	= token of o
A(s,o)	= set of modes

CASE A(s,o) OF

{r}	if subject s can read object o
{w}	if subject s can write object o
(r,w}	if both read and write
0	if neither read nor write

cont(o)	= contents of object o
subj	= active subject

The symbol ⊘ designates the empty set. The subjects and objects are modelled as members of the sets S and O. The 2-dimensional access array A represents the *access matrix* for the system. The state of the system at any given instant is expressed as a set of values of all the state variables *{S,O,acl,tok,cont,subj}*.

Step 2: *Define the secure state*

The definition of the secure state is a mathematical translation of the *properties* (a) and (b) into an *invariant.*

INVARIANT: The system is secure iff

for all $s \in S$, $o \in O$

if $r \in A(s,o)$, then $acl(s) >= tok(o)$

if $w \in A(s,o)$, then $tok(o) >= acl(s)$

The notation $s \in S$ means "s is contained in set S", and the notation $s \notin S$ means "s is not contained in set S".

Step 3: *Define the transition functions*

A *transition function* is defined as any procedure call to a system service routine requested by a subject, with the desired aim of specifically changing a *state variable*. Within the context of the PC LAN implementation, these would be calls to the DOS operating system IO which are vetted by the CSS. The PC LAN DOS CSS implements the following functions:

> **create_object** *(o,c)*;
>
> set_access *(s,o,modes)*;
>
> **change_object** *(o,c)*;
>
> **write_object** *(o,d)*;
>
> **copy_object** *(from,to)*;
>
> **append_data** *(o,d)*.

Define the prime symbol (') in front of a state variable to refer to a state variable in the new state. Unprimed variables refer to the old state.

> **create_object** *(o,c)*
>
> if $o \notin O$
>
> then $'O = O \cup \{o\}$ and
>
> $'tok(o) = c$ and
>
>   for all $s \in S$, $'A(s,o) = 0$ □

```
set_access $(s,o,modes)$
if $s \in S$ and $o \in O$
and if
$\{ [ r \in modes$ and $acl(s) >= tok(o)]$ or $r \notin modes \}$
and
$\{ [ w \in modes$ and $tok(o) >= acl(s)]$ or $w \notin modes \}$
then $'A(s,o) = modes$ $\square$
```

Gasser [GASS,1988] states that functions must be regarded as *atomic*, i.e. that they are indivisible and uninterruptible. Specified state changes happen at once, without the passage of any time 'during' the transition. This is very important when modelling multiprocessor systems with multiple queued requests to the CSS.

Step 4: ***Prove the transition functions***

For each function, we require to prove

Invariant *and* **Function** *imply* **'Invariant**

This is generally accomplished by the application of rules of formal logic, see section 4.6. and Chapter 6.

Step 5: ***Define the initial state***

Mathematically, the initial state is expressed as a set of initial values of all the state variables of the system.

$$\{S_o, O_o, acl_o, tok_o, cont_o, subj_o\}$$

With the framework of a formalism, the initial state is expressed in a set of *axioms*, see section 4.6.

Step 6: ***Prove the initial state***

In order to ensure that the initial state is secure, we must specify *constraints* on the values. Constraints on transitions are also required for several reasons:

1. non-secure transitions: the old and new values of variables must maintain a secure relationship;
2. control on subjects: subjects should not be allowed to invoke certain operations under certain conditions;
3. controls on information: a model that restricts modification of information must control transitions capable of modifying that information.

4.3.3. Non-Secure Transitions

By way of example, we can rewrite the **create_object** function slightly so that it allows the token of an existing object to be changed.

create_object *(o,c)*

'tok(o) = *c*; and

if o $\notin$ O then 'O = O $\cup$ {o}; and

for all s $\in$ S, *'A(s,o)* = *0* $\square$

As before, access to the changed or created object is removed for all subjects, so the function satisfies the invariant. But the function now allows a severe security violation - the possible downgrading of the token of an object.

The reason is that the original security policy as defined in the properties said nothing about the possibility that tokens of objects might change. We therefore need to augment the properties with *constraints*.

Let us specify property (c) as

Property (c): "The token of an object cannot decrease".

Because downgrading involves a state transition, converting property (c) into a mathematical statement requires a *constraint* rather than an *invariant*.

Constraint 1: For all $o \in O$, $'tok(o) >= tok(o)$

This constraint states that the token of an object is only allowed to increase or stay in the same state.

4.3.4. Controls on Subjects

Other constraints on transitions restrict the operations that subjects are allowed to invoke. For example, within the context of the CSS, a constraint is required that prevents subjects from changing the tokens of objects to which they have no access. Indeed, we may wish to restrict this operations only to the *system management* subjects.

Property (d): "A subject may modify another subject's access to an object only if the first subject can read the object"

Constraint 2: For all $o \in O$,

$\quad$ if $\mathbf{r} \notin A(subj,o)$

$\quad$ then for all $s \in S$, $'A(s,o) = A(s,o)$

set_access as specified does not satisfy this constraint as yet, so the proof of security would fail.

4.3.5. Controls on Information

The major limitation of the theory presented so far is that the rules and constraints only change *access rights* to information. They do not control changes to the information itself. This would be satisfactory if the purpose of the model were merely to formalise a security *policy*, rather than the functions of the system itself. But suppose that we wish to model an operation on the data contents of an object, such as

```
write_object (o,d)
    if $o \in O$ and $\mathbf{w} \in A(subj,o)$
    then $'cont(o) = d$ □
```

write_object does not change any variables mentioned in the invariant or in any of the constraints specified, so it is secure according to the model. The model is insufficient because it only expresses the *potential access* of subjects to objects (as represented by the access matrix), and does not consider the information either read or written. We can add the constraint

Constraint 3: for all $o \in O$

$\quad$ if $w \not\in A(subj,o)$

$\quad$ then $'cont(o) = cont(o)$

4.3.6. Information Flow models

One deficiency in the classical proof techniques used for state machine models cannot be solved by adding invariants and constraints. This deficiency involves the flow of information, rather than the control of security attributes of subjects and objects. Consider the following operation

copy_object (from,to)

$\quad$ if $from \in O$ and $to \in O$ and $w \in A(subj,to)$

$\quad$ then $'cont(to) = cont(from)$ $\square$

This function copies the contents of one object into another, if the subject has write access to the destination object. The function is NOT secure because, in failing to check the read access in the *from* object, the *to* object may be written with information to which the subject has no access, and the subject may later read the written object. It has been asserted [GASS,1988] that it is impossible to contrive an invariant or constraint to overcome this.

4.4. Formal Specifications

In the previous section, a brief overview of some mathematical techniques for the definition of a security model of a system were discussed. It is important, however, to distinguish between the process of writing a formal model and the process of writing a formal specification. Formal specifications are required only for systems that must maintain the highest degree of security. The CSS presented in this thesis is such a system.

The purpose of a formal specification is to describe the functional behaviour of the system in a manner which is precise, unambiguous, and amenable to computer processing. The purpose of the computer processing is to carry out various forms of analysis on the specification with the minimum chance of human (or machine) error.

The primary goal of the analysis is to prove properties about the specification. Within the context of formal logic, the analysis is carried out by the proving of *theorems* about security within a logically self-consistent axiomatic framework by means of a *propositional calculus*. Proving that the specification conforms to the functions, invariants and constraints of the model is the first step in the formal *verification* of the system, the final step being the proof that the implementation adheres to the formal specification.

4.5. Methods of Decomposition

The major problem with the choice of a level of formalism is one of extremes. It is possible to write a highly abstract specification which closely resembles the model, but in such instances the task of convincingly demonstrating correspondence between the final code and the specification is enormous. Conversely, it is possible (at least in theory) to produce a specification so detailed that the formalism closely matches the visible operations of the system function for function and parameter for parameter.

Clearly, neither approach is satisfactory in practice; it is necessary, therefore, to devise methods of *decomposition* whereby the formal verification proceeds by stages from the most abstract to the most detailed, each stage being a logical consequence of the previous one.

4.5.1. Data Structure Refinement

The *data structure refinement* method employs a refinement of detail at different levels of abstraction. Each layer of the specification is a state machine that completely describes the system. The topmost layer is highly abstract and combines multiple data types, variables and functions into a number of simpler functions. The second layer adds more detail, dividing generic functions about subjects and objects at the top layer into specific functions about specific types of objects and so forth. Once the second layer is written, it is shown to satisfy the mapped invariants and constraints (and hence the same security properties) as the top layer, and subsequently, the top layer is no longer needed.

Similarly, more detail may be added at each successive lower layer, the mapping correspondence demonstrated, and the higher layer discarded, until the bottom layer is reached. This lowermost layer (closest to the implementation) will correspond very closely to the variables and functions of the code, making a very precise and detailed description of the interface to the system, and a specification from which a number of designers could write systemsacross a number of platforms which would be functionally equivalent.

4.5.2. Algorithmic Refinement

In contrast to the previous technique, whose lowest layer specification presents the *external* view of the system, the *algorithmic refinement* technique allows the specification of some of the internal structure of the system. The approach views the system as a series of layered abstract state machines, each machine making available a set of functions for use by the machine above. The implementation of each function in a machine consists of an abstract program that calls functions in the machine below. The lowest level machine provides the most primitive functions; those that cannot be further decomposed.

The agent concept used in the Comprehensive Security System is similar in approach. The security *functions* are provided by a number of security *services*, sequenced under the control of the Security Services Agent, which in turn calls the hardware and software security *mechanisms* under the control of the mechanism agents.

4.5.3. Procedural Abstraction

The technique of *procedural abstraction* directly models the way a system is implemented, that is, as a set of nested procedure calls. (Although this assumes a pure software implementation, it is also applicable to hardware, where the functionality is controlled by *firmware* in Read Only Memory, for example). As in the algorithmic refinement technique, each function in the model is equivalent to a function in the implementation, but with the important difference that the technique does not require the system to be built in layers. Procedural abstraction describes how a function manipulates its arguments, *not* how the function affects the *global state* of the system. Because the model produced by this technique is so close to the code of the implementation itself, it can be regarded more as a program-proving system than a specification system. Indeed, it is the basis of a PASCAL-like pseudocode, as well as the foundation of formal language tools such as *Z*, developed jointly by Oxford University and the IBM (United Kingdom) Laboratories in an early attempt to deal with formal analysis of large software systems [McMO,1989].

4.6. Methods Using Formal Logic

4.6.1. Introduction to Formal Logic

The concept of a formal logic consists of a *formal language* and a *deductive apparatus*. This provides a way of generating and manipulating abstract strings of symbols. If the system is to be useful, however, the strings in the formal language must be ascribed some meaning, or *semantics*. This is achieved by providing an *interpretation* for the formal system.

A formal language comprises two parts, its alphabet which specifies what symbols are to be found in the language, and its syntax which specifies how these symbols may be put together. An alphabet is specified by writing down the symbols in curly brackets, {...}, separated by commas, with conventional shorthand notations where there are many ordered symbols such as $\{a,b,\ldots,z\}$. This piece of notation is part of a *metalanguage*, used for describing the formal language. By way of example, two alphabets are given which correspond to two common formal languages:

1. A language for expressing the real numbers:
   {0,1,2,3,4,5,6,7,8,9,.}

2. A language for expressing musical notes:
   {A,B,C,D,E,F,G,♯,♭,♮}

It is possible for two languages to have the same alphabet. For example, the alphabet belonging to the formal language used for the section numbering scheme in this thesis is exactly the same as that used for the real numbers in example (1) above. The factor that differentiates the languages is that the syntax of each allows different strings to be formed in the two languages. For example:

1.	Acceptable section numbers:	3.2.	1.2.3.	3.6.1.
2.	Unacceptable section numbers:	.7	3.	.5.
3.	Acceptable real numbers:	23.7	12	6.75
4.	Unacceptable real numbers:	3.6.1		

An acceptable string in a formal language is called a *well formed formula* or *wff* (pronounced 'woof') of that language. The formal language itself is nothing more than just the collection of all its well formed formulae. Exactly what specifies a wff is defined by the syntax (sometimes known as *grammar* in some books) of the language. To give the syntax, a metalanguage is required for expressing the rules. This can be a *natural* language such as English, or another formal language. For example, The language $\mathscr{L}$ has as its alphabet { $*$, ◇ }, and its syntax is given by the rule "A wff in $\mathscr{L}$ is any finite string of zero or more $*$ symbols, followed by between one and four ◇ symbols, or a string of one or more $*$ symbols with no ◇ symbols following".

The following are examples of wffs in $\mathscr{L}$:

$* * * * * * * * ◇ ◇ ◇$

$* * * * ◇ ◇ ◇ ◇$

$◇ ◇$

$* * * * * * * * * * * * * * * * ◇$

$* * * * * * * *$

The following are not wffs in $\mathscr{L}$

$* * * ◇ ◇ ◇ *$

$◇ ◇ ◇ *$

Note that we do not seek at this stage to ask what the wffs *mean*, since no meaning has yet been assigned to the wffs. The important question is whether the given string is a wff of the formal language. If we are using English as the metalanguage, even this may not be easy to answer, because natural language descriptions of things are often very hard to reason about accurately. It is precisely for this reason that computer software specifications written in natural languages are so difficult to verify rigorously. One of the main applications of formal languages is in the exact specification and hence accurate verification of computer systems, see Chapter 6.

The next stage is to give the formal language some *semantics*, by specifying the meaning of each wff admitted by the syntax. This is achieved by giving the language an *interpretation* which assigns a value from some domain of interest to each wff. We say that this value is the *meaning* of the wff under the particular interpretation. To extend the previous example, let,

$*$ denote 5

$\diamond$ denote 1

and let the placing of one symbol next to another indicate that their values under the interpretation are to be added together. So

$\diamond \diamond \diamond$ denotes $1 + 1 + 1 = 3$

$* * * \diamond \diamond$ denotes $5 + 5 + 5 + 1 + 1 = 17$

So the interpretation $\Im_1$ of language $\mathscr{L}$ assigns to each wff of $\mathscr{L}$ a value from the domain 1,2,3,... As a shorthand for the statement 'in the interpretation $\Im_1$ $\diamond \diamond \diamond$ denotes 3' we will write the notation $\Im_1 (\diamond \diamond \diamond) = 3$.

This is, of course, only one interpretation of the language. Had we assigned a different interpretation and put

$\Im_2 (*) = 10$

$\Im_2 (\diamond) = 2$

with the same addition rule as above, then this interpretation $\Im_2$ assigns to each wff in $\mathscr{L}$ a value from the domain 2,4,6,8,....

It is the ability to use pieces of formal notation to denote different things under different interpretations that makes the mathematics so powerful.

Having achieved the stage where a formal language may be defined in terms of meaningless strings of symbols from an alphabet, meaning then being assigned to the strings, we can *describe* things in the formal notation. The final stage is to add to this descriptive ability a facility to manipulate the strings in a formal manner by analysis of their syntactic shape. The consequences of this are extremely powerful because it allows *inference* of facts about the system that were not known at the outset. One of the most powerful features of this idea is that it is not necessary to know the meaning of the strings being manipulated, it is merely sufficient to follow the rules. The mechanism by which inference is performed is called the *deductive apparatus*, and the complete framework is called a *formal system*.

The most important requirement of a deductive apparatus is that it makes *no reference* to any particular interpretation. If it did, the usefulness of the apparatus would be destroyed because the apparatus would be application-specific and the generality would be lost. To give an example, in the formal system called *arithmetic*, it is possible to add two numbers without concern about what they stand for. They could represent apples or cars or even just the abstract notion of 'number'; the interpretation in no way affects the deductive apparatus.

The two components of a deductive apparatus are:

> **AXIOMS** - wffs which can be written down *without reference* to other wffs in the language;

INFERENCE RULES - rules which allow the production of wffs in the language as an *immediate consequence* of other wffs.

It is very important to note that the *axioms* are written down at the start as statements about the system which are *believed* to be true. If they *are* true, then the wffs which are inferred from them will also be true, but since the axioms themselves do not originate from other 'meta-truths' they cannot be guaranteed to be true. Consider the following formal system:

ALPHABET { * , ◇ , o }

GRAMMAR sentence = *string of stars*," ◇ ", *string of stars*, " o ", *string of stars*;

where *string of stars* = " * ", *string of stars*, " * ";

AXIOM * ◇ * o * *

RULE "If $a \diamond b \circ c$ is a given wff, where a,b and c are string of stars, then a ◇ b * o c * is an immediate consequence of it".

So to show that * ◇ * * * * o * * * * * is an immediate consequence of the wff * ◇ * * * o * * * *, we can identify a,b and c.

```
* ◇ * * * o * * * *
a    b        c
```

so that applying the rule we get

```
* ◇ * * * * o * * * * *
a    b        c
```

To give this formal system a possible interpretation, let

$\Im(*) = 1;$

$\Im(*\ *) = 2;$

$\Im(*\ *\ *) = 3;$

etc.

$\Im(\diamond) = +$ (addition);

$\Im(\circ) = =$ (equals);

The sentences in the language now take the semantics $a + b = c$, which can either be true of false. The axiom now takes the interpretation $1 + 1 = 2$, which is true. As a consequence, the inference rule can be thought of as expressing that

IF $a + b = c$ THEN $a + (b + 1) = (c + 1)$

which is a sensible result. Note that this result is only 'correct' because of the meaning ascribed to the symbology. Had the subtraction operation (-) been assigned to $\diamond$ instead of addition (+) then the result would not fit into our intuition regarding arithmetic.

Having laid the groundwork for a formal system, it is now possible to address the concepts of *theorems* and *proofs*. A *proof* in a formal system $\mathcal{F}$ is a finite sequence of wffs in the associated formal language, each of which is either an axiom of $\mathcal{F}$ or an immediate consequence of one or more preceding wffs, as determined by the inference rules of the system. A wff which can be *proved* within $\mathcal{F}$ is called a theorem of $\mathcal{F}$. By definition, all axioms of $\mathcal{F}$ are also theorems of $\mathcal{F}$. Using the alphabet, syntax and interpretation of the previous example, let

THEOREM	* ◇ * * * * o * * * * *	
PROOF	* ◇ * o * *	(axiom)
	* ◇ * * o * * *	(inference rule applied to axiom)
	* ◇ * * * o * * * *	(inference rule applied to previous wff)
	* ◇ * * * * o * * * * *	(inference rule applied to previous wff)
Q.E.D.		

Note that in proving the required result, other theorems have been discovered on the way, namely * ◇ * * o * * * and * ◇ * * * o * * * * . Ascribing the meaning of the previous example, it has been formally proven that, given the truth that 1 + 1 = 2, then 1 + 4 = 5.

4.6.2. Mathematical Proofs of Correctness

Mathematical proofs of correctness build on the ideas of formal logic to produce a formal system known as *predicate calculus*. This formalism allows reasoning about statements which are much more complicated than the simple theorems discussed in the last section. The predicate calculus achieves this by adding more expressive power to the formal language and additional rules to the deductive apparatus. For the benefit if the reader unfamiliar with these ideas, a summary can be found in Annex 4.

4.6.3. The Method of Hantler and King

One interesting early approach to the verification of software by a formal method was described by Hantler and King [HANT,1976]. Their method is a formal system using abstraction at quite a high level to yield a methodology based on 'correctness assertions' about software statements. Of course, their methodology is only an attempt to prove the correctness of *execution* of the program. It cannot verify that what the program is executing is *secure* - the aim of the verification methodology described in Chapter 6. Their method is, however, important in laying the foundations for the verification of the CSS.

They begin by defining a very simple Pascal-like programming language, within the framework of which the software to be verified is written. This corresponds very closely to the *formal system* discussed in detail in the last section. The programming language has an alphabet of permissible statements such as {DECLARE, PROCEDURE, INTEGER, END, ... } and so forth. Having defined the language, the meaning of 'correctness' of procedures is discussed. In particular, the constraints on inputs to a procedure and expected relations between inputs and outputs are expressed as assertions over the program variables. An *input assertion* is of the form

```
ASSUME (<Boolean>);
```

and usually appears immediately after the *PROCEDURE* statement. For example, the input assertion ASSUME ($p_1 > 0$); asserts that the value of variable p_1 is assumed to be positive on entry. By contrast, an *output assertion* is of the form

```
PROVE (<Boolean>);
```

and usually appears immediately before the final *RETURN* statement of the procedure. For example, the output assertion

```
PROVE ((X = Y') & (Y = X'));
```

indicates that the values of the variables have been interchanged. This relationship between inputs and outputs would only be satisfied by a correct interchange procedure. A procedure is said to be *correct*, therefore, if the truth of its input assertion ensures the truth of its output assertion. A simple example is the **ABSOLUTE** function, which is required to return the absolute (i.e. no sign) value of its argument:

```
PROCEDURE ABSOLUTE (X);
    ASSUME (true);
    DECLARE X, Y : INTEGER;
    IF X < 0
        THEN Y ← -X;
        ELSE Y ← X;
    PROVE ((Y = X'|Y = -X') & Y ≥ 0 & X = X');
    RETURN (Y);
END; □
```

In this case, no assumptions need to be made about the argument, so the input assertion is set to *true*. The output assertion must specify that, when the return statement is executed, the value of procedure variable Y is the absolute value of the initial value of parameter X. This example is so simple that the correctness can be verified by inspection. The paper then considers the situation if the procedure is too complex for direct inspection.

Central to the development of the methodology is the notion that a proof of correctness for a program is a proof over *all* program inputs. Clearly, it is impractical actually to test the procedure exhaustively by using all possible inputs as arguments, so the proof must be made with a statement *about* 'all inputs'. This approach involves the mathematical technique discussed in the last section of inventing symbols to represent arbitrary program inputs, and then attempt a formal proof involving those symbols. If no special properties of the symbols (other than those expected to hold for program inputs) are necessary for the proof, then the proof is valid for each *specific* input. This abstraction to the general case is a very powerful feature of the formalism, as demonstrated in the examples of the last section concerning ducks.

The paper proceeds to demonstrate the effect of executing the **ABSOLUTE** procedure with an input value of α. The **ASSUME** statement is always deemed to be true, and so places no constraint on the actual value of α. Proceeding to the **IF** statement, it is now necessary to examine the range of possibilities.

If $\alpha < 0$ then the *IF* test would produce *true* and would proceed to the **THEN** clause. Here, Y becomes the negative of X (i.e. $-\alpha$). Arriving at the **PROVE** statement, it is necessary to show that the present value satisfies $((Y = X' | Y = -X') \& Y \geq 0 \& X = X')$. Since $Y = -\alpha$ and $X = X' = \alpha$, this becomes

$$(-\alpha = \alpha \mid -\alpha = -\alpha) \& -\alpha \geq 0 \& \alpha = \alpha$$

which simplifies to $-\alpha \geq 0$ or more simply $\alpha \leq 0$. Therefore, establishing the truth of the **PROVE** statement reduces to showing that $\alpha \leq 0$. But by assumption, $\alpha < 0$ so the proof is trivial, and for the case $\alpha < 0$ the program functionality is correct.

If $\alpha \geq 0$ then the *IF* test would produce *false* and would proceed to the **ELSE** clause. Here, Y becomes the negative of X or α. Arriving at the **PROVE** statement, it is necessary to show that ((Y = X' | Y = -X') & Y ≥ 0 & X = X') is *true* when Y = α, X = X' = α. That is, to show that

$$(\alpha = \alpha \mid \alpha = -\alpha) \ \& \ \alpha \geq 0 \ \& \ \alpha = \alpha$$

or simply $\alpha \geq 0$ is *true*. Again, the proof is trivial since by assumption $\alpha \geq 0$ and again the program functionality is correct. By the nature of the **IF** statement, these two cases are exhaustive and therefore the program is correct.

This analysis may be expressed symbolically in a *symbolic execution tree*, see Figure 4.1.

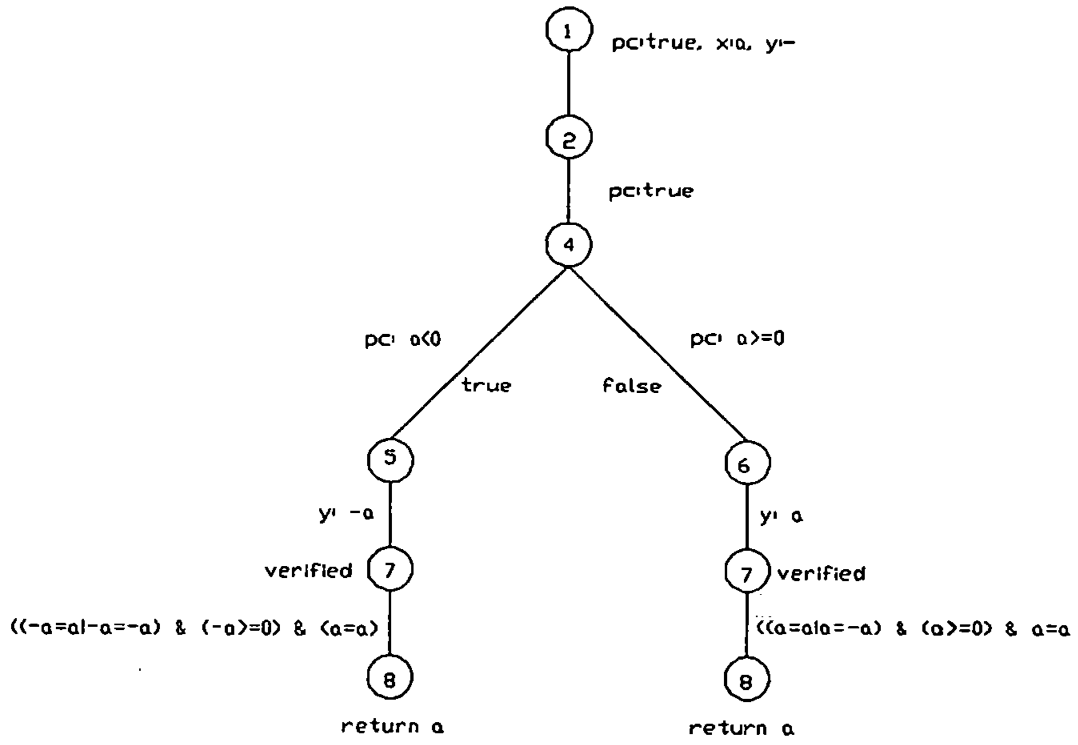


Figure 4.1. Symbolic Execution Tree for ABSOLUTE Procedure

The numbering of the nodes corresponds to the line numbers in the program, and the abbreviation *pc* stands for *path condition*. This is the collective set of values taken by the parameters as execution progresses symbolically down the tree. It is possible for a symbolic execution tree to be either *finite* or *infinite*. Clearly, in the case of an infinite tree, the symbolic execution never stops, and so the final conditions cannot be tested. It is therefore important to construct the program so that the symbolic execution tree will be finite.

4.6.4. A Formal Proof of Euclid's Algorithm

The method is now applied to a formal verification of Euclid's algorithm, an important building block in the cryptographic mechanisms to be described later in the thesis. Euclid's algorithm, which is used to determine the *greatest common divisor* of two numbers, may be written in the formalism as follows

```
PROCEDURE GCD (M,N);
    ASSUME (M>0 & N>0);
    DECLARE M,N,A,B INTEGER;
    A ← M;
    B ← N;
    DO WHILE (A ≠ B);
        IF A > B
            THEN A ← A - B;
            ELSE B ← B - A;
    END;
    PROVE (A = (M,N));
    RETURN (A);
END;
```

We now assign the input assertion. Note that it is possible to assign the value *false* to the **ASSUME** statement. In this case, the symbolic execution tree is *finite* (indeed, it is *empty*), but guaranteed to be correct! As this case is of no interest, let the **ASSUME** statement be *true*. The symbolic execution tree is now non-empty, but infinite in extent, see Figure 4.2.

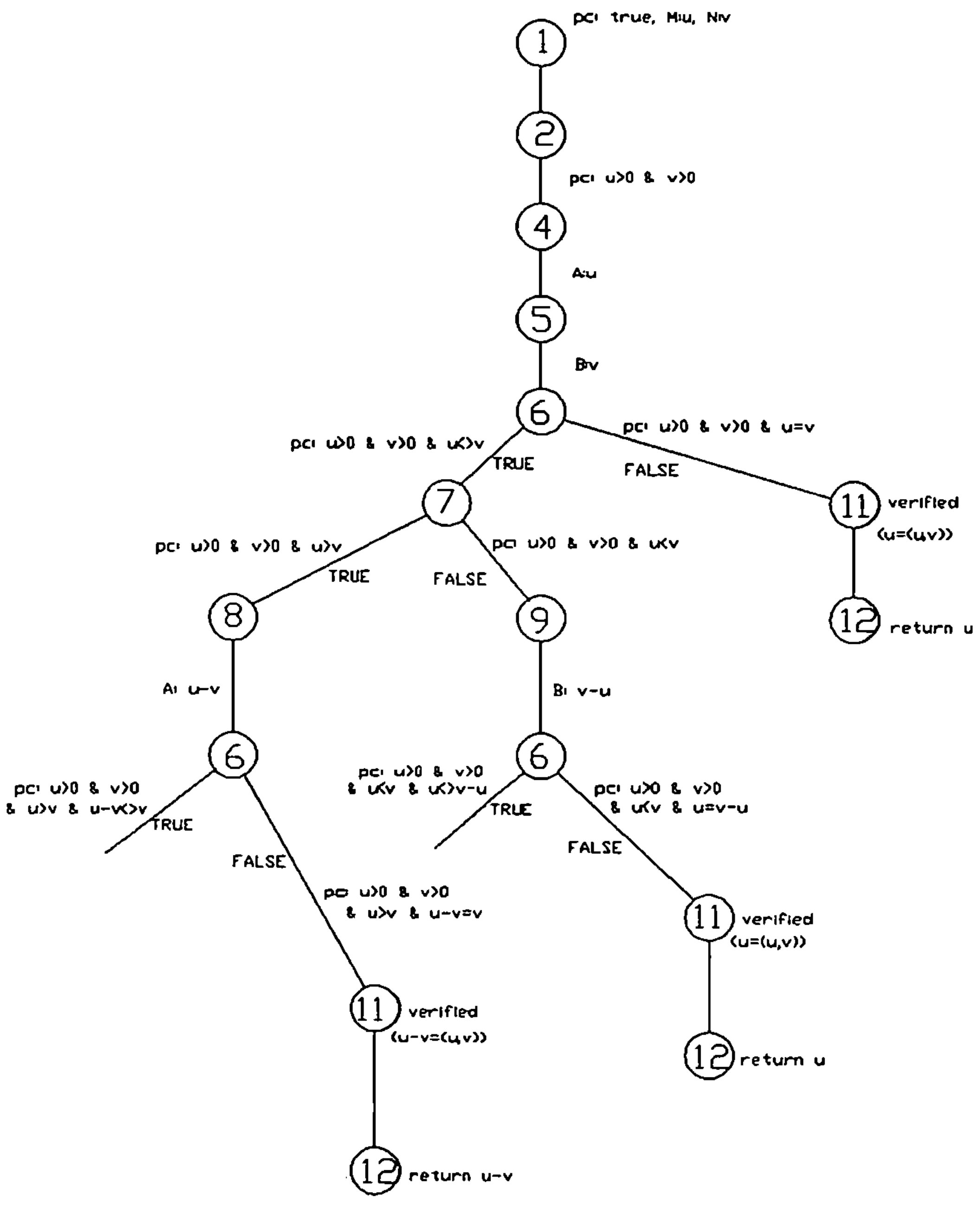


Figure 4.2. Symbolic Execution Tree for Euclid's Algorithm

The reason for the infinite extent of the symbolic execution tree is due to the **DO WHILE** statement. Unless additional constraints are placed on the input values, for example, one made a fixed multiple *k* of the other, there is no way to determine how

many times the **DO WHILE** statement will loop. The imposition of the condition mentioned would constrain the system to k loops, but would destroy the generality of the analysis. The solution to the problem is similar to that employed in the analysis of closed loop transfer functions in control theory. The loop
is *broken* at some point, and if the point chosen is not an **ASSUME** statement, an **ASSERT** statement is introduced to make additional assumptions about the system on exit from the loop. The resulting symbolic execution tree is called a *cut tree*. Applying this to the previous example, we can insert two cuts at line 2, where an **ASSUME** statement already exists, and at line 7, where no **ASSUME** statement exists and so an **ASSERT** statement is required:

```
1    PROCEDURE GCD (M,N);
2    $cut_2$   ASSUME (M>0 & N>0);
3              DECLARE M,N,A,B INTEGER;
4              A ← M;
5              B ← N;
6              DO WHILE (A ≠ B);
7    $cut_7$   ASSERT ((A,B) = (M,N) & A ≠ B);
8                     IF A > B
9                            THEN A ← A - B;
10                           ELSE B ← B - A;
11             END;
12             PROVE (A = (M,N));
13             RETURN (A);
14   END;
```

The cut trees for these cut points are shown in Figure 4.3. and Figure 4.4. respectively.

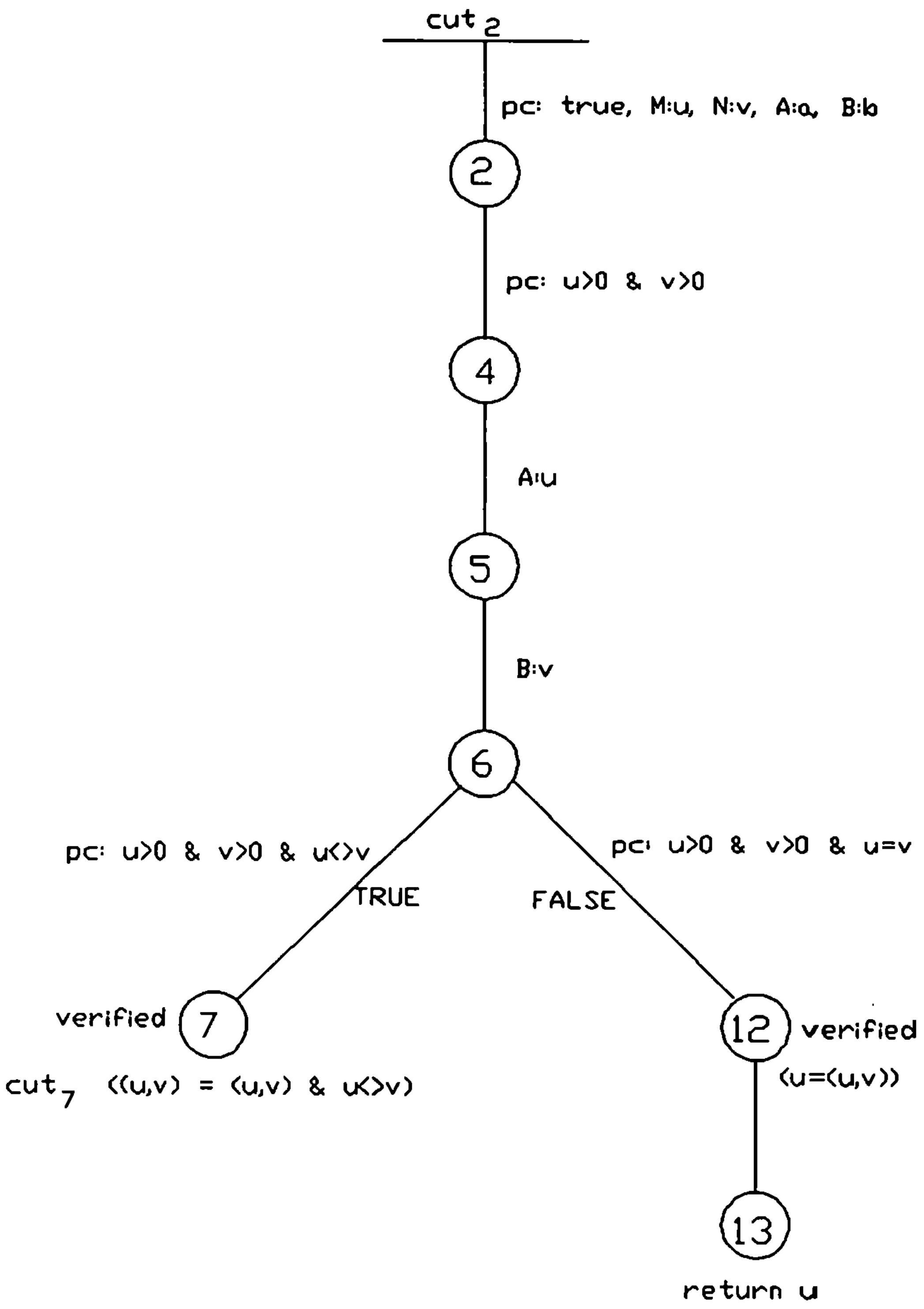


Figure 4.3. Cut Tree for Euclid's Algorithm at cut_2

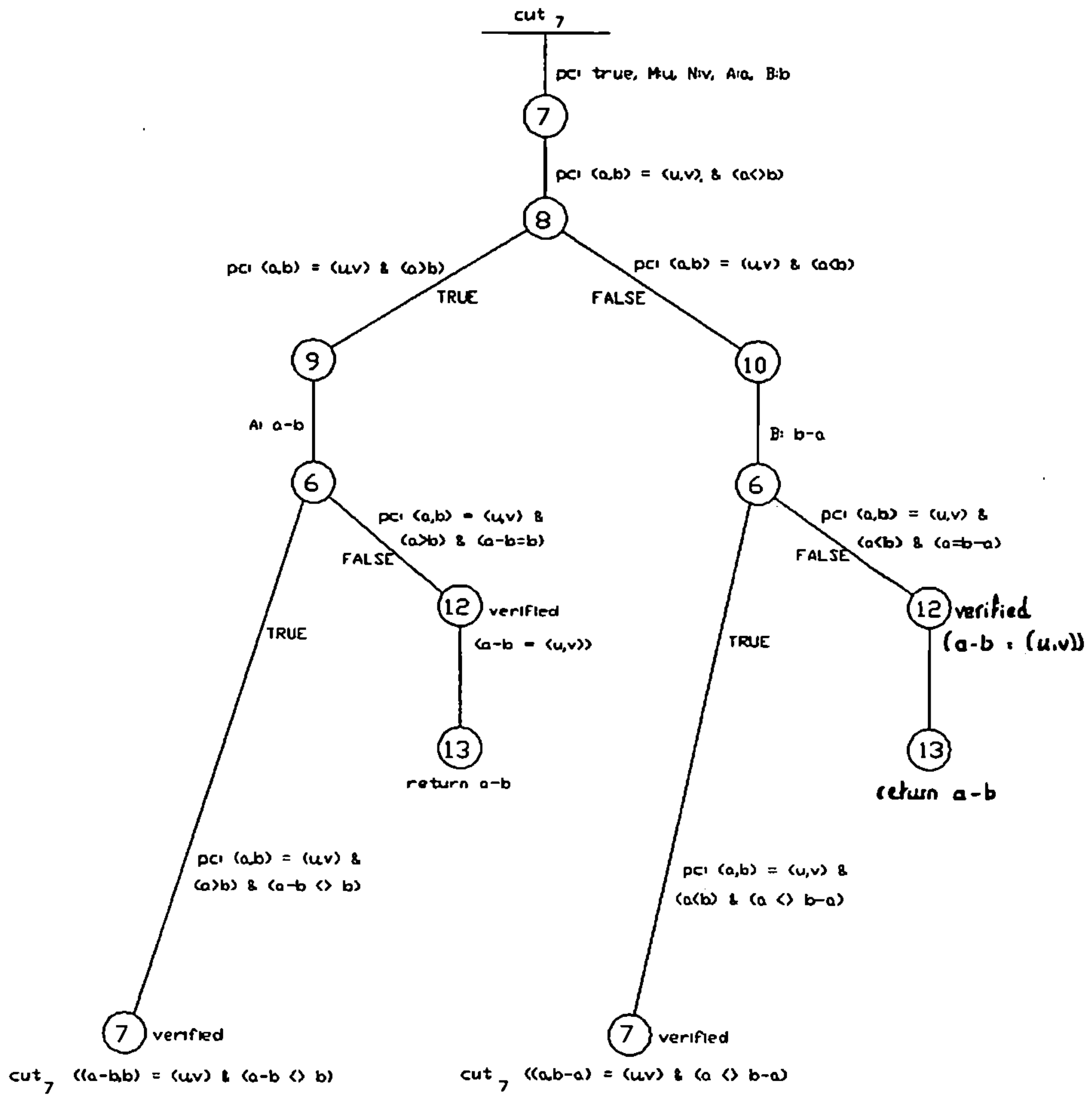

cut 7
pc: true, M:u, N:v, A:a, B:b
7
pc: (a,b) = (u,v), & (a<>b)
8
pc: (a,b) = (u,v) & (a>b)
TRUE
FALSE
pc: (a,b) = (u,v) & (a<b)
9
10
A: a-b
B: b-a
6
6
pc: (a,b) = (u,v) & (a>b) & (a-b=b)
FALSE
12 verified (a-b = (u,v))
13
return a-b
TRUE
pc: (a,b) = (u,v) & (a>b) & (a-b <> b)
7 verified
cut7 ((a-b,b) = (u,v) & (a-b <> b)
pc: (a,b) = (u,v) & (a<b) & (a=b-a)
FALSE
12 verified (a-b : (u,v))
13
return a-b
TRUE
pc: (a,b) = (u,v) & (a<b) & (a <> b-a)
7 verified
cut 7 ((a,b-a) = (u,v) & (a <> b-a)

Figure 4.4. Cut Tree for Euclid's Algorithm at cut_7

Note how the cut prevents the tree becoming infinite in extent - as soon as a cut point is reached, the tree terminates and the path conditions can be evaluated at that point.

4.7. Limitations of Conventional Debugging Techniques

Debugging is the conventional method of looking for faults or 'bugs' in software by stepping through the code using a software tool known as a *debugger*. Generally, a debugger allows the execution of the code one line or instruction at a time, and the user can examine the values of variables and parameters after each step to see where the errors are occurring. It is generally a very painstaking and time-intensive task, even if the code is *linear*, that is, has no loops or recursions. Pure linear code is very rare indeed in real applications, and the existence of loops and recursion can make conventional debugging almost impossible, as at each branch point a decision has to be taken as to which path to follow. When that selected path has been traced, the alternatives also have to be explored. Consider the following short module of recursive pseudo-code:

```
        PROCEDURE A; FORWARD;

1       PROCEDURE C ();
2               CALL PROCEDURE_A;
3       END;

4       PROCEDURE D ();
5               CALL PROCEDURE_A;
6       END;

7       PROCEDURE E ();
8               CALL PROCEDURE_A;
9       END;
```

```
PROCEDURE B (M);
    CASE N OF
    1 : CALL PROCEDURE_C;
    2 : CALL PROCEDURE_D;
    1 : CALL PROCEDURE_E;
END;

PROCEDURE A (N);
    FOR I := 1 TO 12 DO
        CASE N OF
        1 : CALL PROCEDURE_C;
        2 : CALL PROCEDURE_B;
        1 : CALL PROCEDURE_E;
    END;
END;
```

The paths through the module are illustrated schematically in Figure 4.5.

This is a very simple piece of code, and with only 12 iterations through the module, the number of unique paths through the code is over *3 trillion*. Clearly, following each path to look for possible sources of error is out of the question. As demonstrated in section 4.6.3., the requirement to debug can be largely eliminated if more powerful mathematical techniques are applied instead.

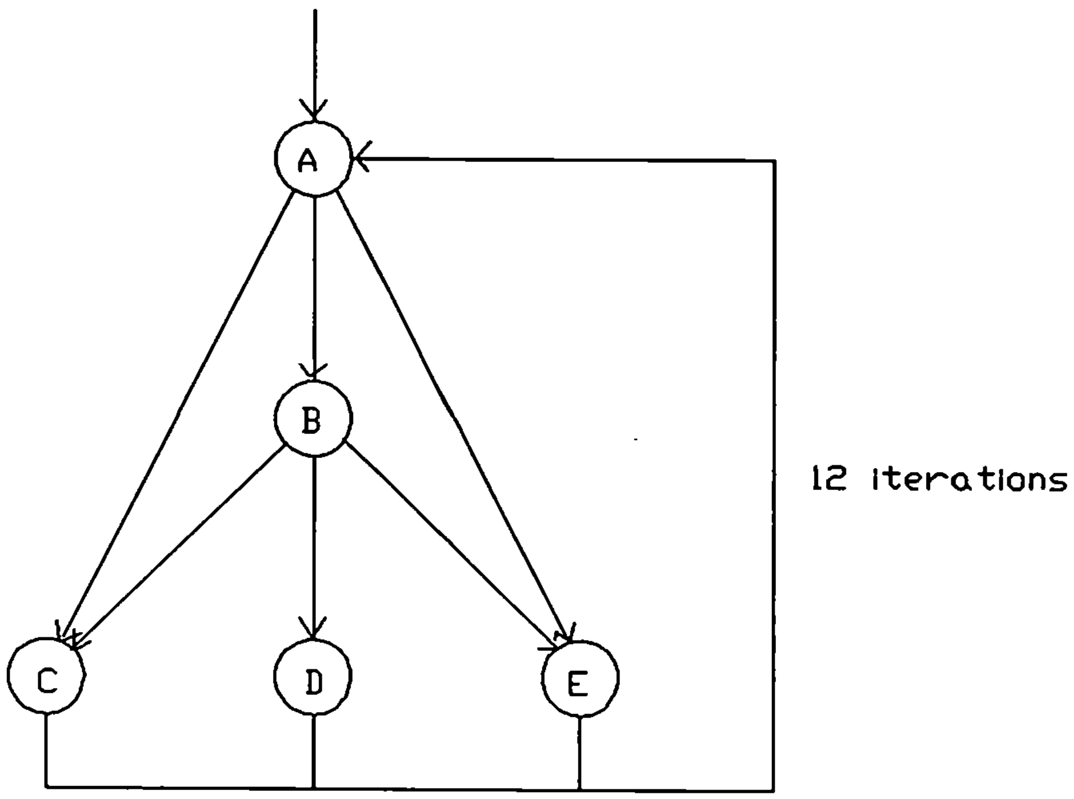


Figure 4.5. Limitations of Conventional Debugging

4.8. Limitations on Current Analysis Methods

The software analysis and verification methods described in this chapter deal with the *semantic* correctness of software and, as applied to security system software, are very important in the verification of the code. They do nothing, however, to ensure the *functional* correctness of the system. Clearly, the difficulties in verifying the former alone are fraught, and to ensure the latter as well by similar means would be very difficult indeed. The major limitation of the current analysis methodologies described is one of scale. The methods are well suited to small-scale analyses of code fragments where the variables, which together comprise the *global state* of the system, may be specifically enumerated. By way of example, consider a very simple system with one

Boolean variable *F*. The variable *F* may by definition assume only one of two possible states, *true* or *false*. Because *F* is the only system variable, clearly the global state of the system is also constrained to one of two states. If the system is now extended to two boolean variables, *F* and *G*, where each may assume two states, the number of possibilities for the global state is now four:

F = false	AND	***G*** = false
F = true	AND	***G*** = false
F = false	AND	***G*** = true
F = true	AND	***G*** = true

If a *byte* variable is now added, taking any value from a range of 256 possibilities (0..255) then the number of global states expands to 1024. As more variables are added, especially those with very large possible ranges (or even those with virtually infinite possibilities such as *reals*) then the number of possible global states rapidly increases past the point where it is no longer feasible to enumerate them explicitly. For a system such as the CSS, there are several hundred variables. Although each variable is carefully declared with the smallest range compatible with functionality (that is, if a variable only requires say 200 different states, it is declared as a *byte* (0..255) as opposed to a *word* (0..65535), the number of possible global states is enormous. The important point about these states, however, is that the vast majority should never arise in the normal functioning of the system, although there is generally nothing actually to prevent them from doing so. This is the source of the software *bug* - the system has taken on a state which the programmer did not intend should arise (or was not even aware *could* arise!). The aim of well written software, especially in the commercial arena, is to include sufficient *error detection* and *fault recovery* routines to detect possible situations where the system attempts to take on a state outside the small subset of permissible states, and return the system to one of the permitted states. Failure to do this will often result in the system 'hanging up', a common criticism of early or poorly written software. For example, a routine to read in a numeric argument from the user (via the keyboard, say) should include error checking to ensure that the input string contains no non-numeric characters. If this check were not present, meaningless data could be presented to a function which expects only numeric arguments, causing the system to enter an unknown state. Of course, the check will not detect the entry of syntactically valid but contextually incorrect numerical data.

In the vast majority of systems, the number of *permissible* states is a very small subset of the number of *possible* states. The vast majority of the unwanted states will probably result in random and harmless system crashes, although in some time-critical applications loss of processing time may be considered to be harmful. While this is also true for a security system, it may also be possible for the system to end up in a state where security is compromised. Although the probability of this happening purely by chance is small, the possibility of an attack being devised specifically to bring this situation about cannot be dismissed.

By way of example, consider the limitations of the analysis described in section 4.6.3. If this is applied to a statement which takes the values *true* or *false*, the symbolic execution tree branches into two at that point. If it were applied to a system containing a CASE statement on a *word* variable, however, the symbolic execution tree would expand into a possible 65536 new branches. Clearly, the analysis flounders due to the number of possibilities, many of which may be redundant. If it is possible to produce a system whose entire design philosophy is based upon careful constraint of the global state to a carefully selected and rigorously analysed subset of the large number of possible states, then a formal analysis based on the permitted subset may be possible.

The *Comprehensive Security System* to be described (in Chapter 5) is an attempt to implement a security system based on the above design philosophy. Furthermore, due to the rigorous design methodology, formal analysis is further simplified by a *reduction technique* which can further limit the number of possible global states which need to be considered in the formal analysis (see Chapter 6).

5. A NEW APPROACH

5.1. <u>A Comprehensive Security System</u>

5.1.1. Security Policy

Within an Open Distributed Processing environment, involving the transfer of information between remote end-user systems, the provision of a generic security function can be conceived in terms of a security policy, rigorously enforced upon those entities who are subject to that policy. The security policy represents the overall set of measures adopted to fulfil the desired security function and covers every aspect of the business of implementing an effective security system. It will involve:

1. provision of physical, hardware and software security mechanisms, such as locked and guarded buildings, protected terminals, encryption and so on;

2. definition of protocols for all data transfers within the system, either embedded within existing OSI protocols, or interfaced to them;

3. enforcement of the fundamental principles of access control, user identification/authentication;

4. provision for effective system resource protection and optimisation of use. This includes such measures as Integrity of Resources, Confidentiality of Use of Resources, Assurance of Service, Accountability of Usage of Resources, Audit Trail etc;

5. provision for monitoring, logging and analysis of the security system at all times, for both optimisation of system resources, and detection of possible subversive activity.

The security policy is formulated and dictated by an *authority*, which is ultimately responsible for the overall performance and effectiveness of the system.

5.1.2. Security Domains and their Administration

Before describing the CSS model, we define some of the terms used. These terms have *precise* meaning within the context of the CSS and cannot be used loosely or interchangeably. Our definitions have been dictated by the security considerations of the architecture. While every effort has been made to ensure that they coincide with similar definitions used in the OSIRM, this has not always been possible. Where ambiguity has arisen, this is due mainly to conflicting requirements or the vagueness and lack of rigour with which the definitions in the OSIRM are made and applied.

The *authority* delegates the implementation of security policy to a system *administrator*. (Some attributes of an administrator correspond with those of a system *manager*). In a large network, there may be a number of administrators responsible for rigid observation of the security policy. The purview of a security administrator is known as a security *domain.* A security domain is defined as a bounded group of security objects and security subjects to which applies a single security policy implemented by a security administrator.

The security domain is a managerial/control concept that defines the scope of a particular security policy. Where the number of security subjects and objects is large, they may be formed into subgroups for ease of management [DTI,1989]. Such a sub-group is referred to as a *sub-domain.* Normally, the policy of the overall domain will apply to all sub-domains. Thus, a domain covers all or part of a given distributed system.

One authority will dictate policy for one domain, and another authority will dictate policy for another domain. A successful association should only be possible if the security policies, services and mechanisms of both end systems are compatible. Although there is no logical difference between local activities and remote activities, a local activity may be assured of compatibility within a security policy local to the domain, whereas a remote activity may require inter-domain 'translation' protocols to ensure effectiveness of an overall security policy, especially when operating as a *mutually suspicious system*, see section 2.9. This may lead to incompatibility between domains. In this event, the incompatibility is arbitrated and resolved by reference to a higher authority. These higher authorities may take the form of regional and then national committees, that must meet given codes of practice, contractual specifications, or the ISO standards. Any authority dictating policy, not conforming to these standards will by default exclude itself from connectivity within the complete open security framework.

Within each domain, the security administrator is responsible for the implementation of the domain security policy and for assuring its continued effectiveness. This responsibility includes the installation of trusted hardware and software, hardware and software functionality, monitoring day to day operations, and recovery in case of breach of security or fault conditions.

A logical model can be constructed with a defined hierarchy where each entity within the model will have specific tasks to perform under the purview of its superior. Within this model, any user entity or application entity that is allowed by the security policy to access the security services can obtain/provide information securely to other authorised users within the ODP environment. A user entity may request the access of an object or service in normal (insecure) mode either accidentally or intentionally, but if this object or entity is itself subject to the security policy, then that policy will force the security services to be invoked for this activity, or access will not be possible at all. This approach maximises the system's ability to account for both human error and attempts at criminal misuse of the system.

5.1.3. Conceptual Model of the CSS Processor

The CSS is conceptualised as a collection of communicating and cooperating *agents*. An agent is defined to be a logical component of the security system, designed to implement a particular function or group of functions. These functions are combined in order to provide security *services*. The agents are independent of each other so that any combination of agents may be used. The same set of agents may provide various services by being combined together in different orders. In order for agents to cooperate they must communicate. Cooperating agents send messages to each other in the form of *protocols*. The component agents of the CSS are carefully chosen along lines similar to the criteria for the choice of layers in the OSI model. The criteria for the choice and number of agents are:

1. not to create so many agents as to make the systems engineering task of describing and integrating the agents more difficult than necessary;
2. to create a boundary where the description of services can be small and the number of boundary interactions minimised;
3. to create separate agents to handle functions that are manifestly different;
4. to collect similar functions in the same agent;
5. to create agents of easily localised functions;
6. to create boundaries between agents where at some time the interface may be standardised;
7. to create an agent where there is a need for a different level of abstraction in the handling of the data;
8. to allow changes of functions or protocols without affecting other agents.

Essentially, the CSS coexists with the application entities it is to protect. Within the context of the Open Systems Interconnection model, the CSS will reside within the Application Layer as another application entity. The CSS has access to both the calling application and the application user to request information when required and these in turn have access to the CSS to invoke functions when required.

The CSS offers the application or user entity a number of security services, which the entity must access through a standard Application Program Interface (API) to the CSS. The API is provided by the CSS and generally depends on the system environment upon which the CSS is hosted. Application entities must provide the correct format of data to meet the requirements of this interface in order to take advantage of the CSS as a value added service. The interface consists of a set of service and associated parameters used by the application and CSS.

While it is considerably easier from a conceptual point of view to imagine the CSS purely in local terms, it is important to remember that the CSS as a whole is a *distributed* entity. While this is no way affects the logical operation of the agents, they cannot be considered as being 'in one place'. It is possible that part of an agent may reside on one processor and another (or even duplicated) part may reside on another. In a fully distributed implementation, components of agents will be duplicated many times. This is similar to the OSI application FTAM described in Chapter 3 where part of the application resides on the initiating end system and another part on the responding end system. Neither can work without the other, but the physical displacement of the components in no way affects the logical functionality.

In practice, the *local* components of the CSS may comprise a trusted, tamper-proof hardware module, and associated software [WEIN,1987]. Such an implementation would give a very significant improvement in physical security over the equivalent software-only system, especially with regard to the security of locally held SMIB data, upon which the integrity of the system critically depends. The protection of local agents within the module also eliminates the requirement for encrypted protocols between them due to the inaccessibility of the internal data bus. It is straightforward to construct a module using battery-backed RAM for the SMIB, which is powered down on detection of intrusion, destroying the data in the SMIB. This would prevent the compromise of the users' secret keys and other data which would allow an assailant to defeat the system. Of course, the CSS as a whole is a distributed application entity, and so those agents which are remote from each other must use secure protocols for communication.

The conceptual model of the CSS is shown schematically in Figure 5.1.

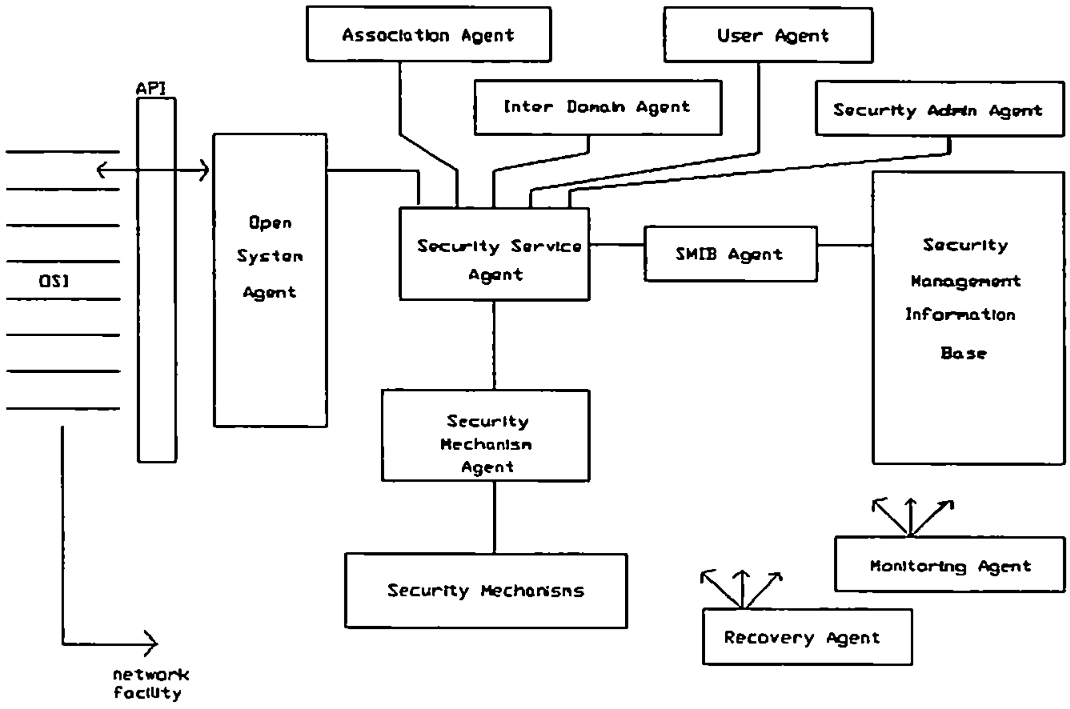


Figure 5.1. Conceptual Model of the CSS

5.1.4. Security Services Supported by the CSS

Although the CSS is implemented as an application entity, it offers a full OSI-wide flexibility due to the interface architecture. It is very important both from the conceptual and practical points of view, to appreciate the 'vertical' structure of the proposed interface. An advantage of this system is the potential flexibility due to the possibility of the CSS functions being called by other than operations in the Application Layer. The CSS API, which could take the form of a software interrupt, for example, is accessible from any of the OSI layers, not just the Application Layer. It is quite permissible for the Transport Layer, for example, to request data encryption services from the CSS. This conforms with the recommendations of OSI, which states that while the majority of security functions can be carried out at the Application Layer, there are a few which may need to be implemented in different layers.

The CSS supports, among others, the following security functions:

1. invocation;
2. identification;
3. authentication;
4. key generation;
5. key distribution;
6. encryption;
7. decryption;
8. signature;
9. verification;

but the full range of OSI recommendations in [ISO,7498-2] should be possible.

In summary, many different security needs can be met by the concept of a common set of security agents provided externally to the application processes. The functional modularisation of the system in this manner makes possible the general definition of a flexible security architecture. These agents will be involved with the interactions between users and applications, and the interaction of applications. These agents, their interaction and management are central to the concept of the Comprehensive Security System to be described. In the practical realisation of the CSS concept, a security model comprising *ten* such agents is suggested.

5.2. The Agents of the CSS

5.2.1. User Agent (UA)

Since the CSS can select which interface will be utilised, either the Application interface or the CSS user interface itself (dependent on the state and requirements of both the user and the CSS), the User Agent (UA) of the CSS comprises 'half' the interface to the CSS as seen by the user entity. This will occur when the application has requested a security service from the CSS, and the CSS requires some information directly from the user, such as a password, etc. The main functions of the User Agent are:

1. to interface between the user entity and the Security Service Agent (SSA) of the CSS;

2. to maintain a library of user entity request statements, via which the UA will determine request/response validity, and suitable responses to the user entity according to a strict set of rules, thus limiting the number of possible user actions;

3. to interpret all data from the user entity and ensure its validity before presenting it to the Security Services Agent (SSA), and to determine the location and nature of errors, and inform the SAA accordingly. Also, to process all data from the SSA into a form suitable for interpretation by the user entity before presentation to the user entity;

4. to accept a request from the SSA when the service requested requires further information from the user, and to act upon this to interrogate the

user in a suitable manner for this information.

The UA is conceived as a separate entity from the SSA because the UA must be capable of interfacing with many user entities, thus freeing the SSA from the complexities of multiple interfaces.

5.2.2. Security Services Agent (SSA)

The CSS Security Service Agent (SSA) is the central control agent of the CSS. It is responsible for:

1. accepting and checking the validity of all CSS service requests from the External Environment Agent (EEA). This is an important function which is necessary to prevent invalid calls from trying to confuse or subvert the CSS into performing functions which are not permitted by the security policy. This validation is accomplished by the SSA checking all requests for security services against the user entity capabilities and privileges stored in the SMIB and the logical state of the sequence of operations carried out to that time (see Chapter 6). Any request not expressly permitted for that user entity by the SMIB will be refused;

2. selecting the appropriate service mechanisms pertinent to the function to be performed under the supervision of the SMIB via the SMIB Agent (SMIBA), passing the relevant sub-function control data to the service mechanism agents, and sequencing the service mechanism agents correctly to perform the requested security service;

3. ensuring the correct routing of the information data to and from, and in the correct sequence among, the security mechanisms. It is possible to implement two completely different security services with the same set of security mechanisms by merely using them in a different order. For example, two security mechanisms implementing a compression (hash) function such as DES in block chaining mode, and an RSA encryption/decryption scheme could be used for (1) a hybrid file encryption system for a confidentiality service, (2) file and message signatures for non-repudiation and integrity checks and (3) checking signatures for authenticity and integrity ;

4. checking with the SMIB the capability of the user entity, and determining whether the user entity has the privilege to execute the requested service;

5. switching between Application UA and CSS UA.

5.2.3. ~~Service~~ Security Mechanism Agent (SMA)

The Service Mechanism Agent will:

1. accept control commands from the SSA, and select and control the security mechanisms to perform the service requested, including sub-function selection of multi-function mechanisms. For example, a DES card could perform normal block encryption, block chaining mode encryption and so on;

2. return to the service agent status information including details of function performed, status of operation (success, failure) and other data resulting from the process.

5.2.4. SMIB Agent & SMIB (SMIBA)

The Security Management Information Base (SMIB) is the 'heart' of the CSS and is the most important unit from the security point of view, and must be protected to the highest level of security. The SMIB will comprise the repository in which the CSS maintains all data pertinent to the security function, including such data as identifications of authorised users, authentication data, user entity capabilities and privileges, and so forth. The SMIB Agent (SMIBA) will be responsible for interfacing the SMIB to the other CSS agents, including accepting and processing all requests from the service agent for information from the SMIB, including such data as user entity identity checks, user entity authorisation, user entity capabilities and privileges, object entity validity, object entity authorisation, object entity security status.

The SMIB holds the following information:

System User Entity Data

This comprises of information concerning the users' rights to access the system and their capabilities and privileges within it and includes:

1. a user system-wide name (unique identifier);

2. a hashed version of the user entity's ~~entities~~ password, for use during authentication. The user entity therefore enters his password, which will be hashed via a one-way function before comparison. This renders compromise of the SMIB username/password file useless, as it is computationally infeasible to invert the hash function and hence to reconstruct the password;

3. further data known only to the user that may be used for further authentication in the case of highly privileged operations or possible uncertainty of identity (this may be used in a future semi-intelligent system which takes account of users habits. If a user habitually uses the system only in the mornings, then suddenly uses the system one night, the CSS may require further proofof identity than just the normal password);

4. a capability token summarising the users' rights and privileges to perform certain operations. This token is passed to the SSA when it needs to validate a user request.

Security Function Sequencing Data

As the CSS can perform a number of functions with a limited number of security mechanisms, in order to ensure that security services are correctly performed in accordance with security policy, the SSA will control the Security Mechanism Agents strictly in accordance with sequencing data held in the SMIB. This will prevent irregular requests (which may have been specifically constructed ambiguously as an attack on the system) from being accepted by the SSA. Details of the sequencing operation are given in Chapter 6.

Temporary Data Store

For a full-duplex connection, or with a multi-user arrangement associated with the encryption functions, chained or feedback vectors will be required for the two different transmission directions, and possibly for a number of simultaneous connection conditions. This information must be stored until the relevant data arrives. Also, it is possible that the CSS will be interrupted from processing I/O to service a request from another local user entity. Should the CSS be about to encrypt a sensitive piece of data which is still in plaintext, it will store this data within the SMIB to ensure that it is safe until the CSS has time to process it. Similarly, incoming data may also be stored here until it is processed.

System Objects Data

The SMIB is likely to hold data on the general security status of objects within the system (files, databases etc) in the form of *tokens*. When a user subject entity wishes to access a system object entity, the user entity is authenticated against his SMIB data as described. The capability of the user entity is also matched against the classification of the object entity to ensure that the user entity privilege is equal or higher.

5.2.5. Security Administrator Agent (SAA)

The Security Administrator Agent (SAA) is responsible for allowing only the administrator to provide modifications to the existing system. There must be very strict protocols and authentication for this type of operation within the system. The SAA is also responsible for the strict imposition of system security policy upon the individual operation of and interaction between, the other agents of the CSS. The main function of the SAA involves controlling the SMIBA to place information into the SMIB, or modify existing information as new users are added to the system, existing users removed, security policy updated, user capabilities and privileges modified, and adding/modifying mechanisms and services.

5.2.6. External Environment Agent (EEA)

The External Environment Agent is responsible for:

1. accepting security service requests across the API and interpreting, validating and routing the requests to the Security Service Agent. Application software packages without the necessary API will not be able to call the CSS in the first place. Those packages with the API which make CSS request calls in error will be returned an appropriate error code by the EEA. (See example under SSA for the likely types of information and control data to be distributed);

2. ensuring that all output from the CSS, including control and information data, is routed back across the API into the same layer which originated the request and the control/information data, to prevent 'short-circuiting' of layers.

It should be noted that this agent could be specific to, say, an OSI implementation of the CSS and this is reflected in Figure 5.1. where the EEA is referred to as the *Open Systems Agent*. In general, however, the agent will provide the interface to whatever underlying network architecture is in use.

5.2.7. Monitoring Agent (MA)

The primary task of the Monitoring Agent (MA) is ensuring that the sequencing of the state-machines that generate the security mechanisms and protocols, under the control of the SSA and SMA, is correctly carried out. The MA is also responsible for accepting and processing all data gathered by the SSA and recording it, including such data as security service requested, date and time, calling user id, calling process, status (success failure) etc. It is envisaged that the MA will itself internally request one of the CSS encryption services, to encrypt the log ready for storage. The only entity with access to the log will be designated levels of system administration, who will possess the decrypting keys allowing managerial access to the log for the preparation of audit reports for security management and resource optimisation purposes. Such access rights will be stored in the SMIB.

The MA could be an AI-based module that will detect problems and even likely problems before they occur, and take the necessary action for preventative or remedial measures.

5.2.8. Recovery Agent (RA)

The Recovery Agent (RA) is responsible for all system fault protection and CSS error recovery. Faults and errors may be caused either by hardware failure of units both within the CSS and external to it, and also by certain combinations of situations with which the CSS cannot cope, due to ambiguity of requests for example. The RA will perform the important task of detecting these errors, and placing the CSS into such a state as to maintain the security integrity of the system, so that the CSS is not left in a state where it is vulnerable to attack.

Faults outside the CSS could in certain circumstances also produce system errors. For example, an incorrectly constructed or incomplete data structure could be ambiguous, and the CSS may 'hang'. Internal error recovery routines will automatically re-request the data, but in the absence of response, the CSS will place itself into a stable, secure state. The CSS has in effect, recovered 'internally' from the error, but cannot, of course, influence events outside the CSS. ('Inside' and 'outside' the CSS refer to software modules within the CSS kernel and those outside the kernel respectively.) Obviously, in a distributed security system, the boundaries cannot be clearly defined. The external error must be recovered by the run-time library of the application package.

In the case of a fault developing with either the SMIB or the SMIBA, the CSS protocols must be designed such that the CSS Recovery Agent (RA) always returns fatal errors to the EEA for ALL requests. Thus, failure of the SMIB terminates all security activity on the local terminal. This differs from faults which may develop with other CSS components, which will not return severe errors, but merely limit the operational effectiveness of the system to those functions not requiring the damaged facility.

5.2.9. Association Agent (AA)

The Association Agent (AA) is responsible for the security control of the association between remote end user entities throughout the duration of the connection. It is responsible for sending the appropriate data when the connection is set up, such as keys, vectors, time stamps and so on, for exercising supervisory control during the connection, and for clearing down the association from the security facility aspects. In addition, detection of denial of service attacks would be possible with this agent by the sending and receipt of random supervisory packets, subject to the current quality of service conditions.

5.2.10. Inter-Domain Communication Agent (IDCA)

The connectivity of entities within a domain is assured, as all communicating entities are subject to a common security policy. Inter-domain communication, however, presents a special problem. The communicating entities in the two domains require the use of 'translation' protocols to ensure a seamless continuity of security around the association. The Inter-Domain Communication Agent (IDCA) is responsible for recognising inter-domain associations, and invoking additional protocols as necessary. Inter-domain translation may or may not be possible.

For example, the remote entity may be using a form of encryption unknown to the local entities CSS processor. In this case, translation is impossible. The IDCA will note that decryption by the local security mechanisms is not possible, and will flag the appropriate error. The RA will return the appropriate flags to the calling application, which will then either terminate the association, or re-request the remote entity to communicate in another secure mode.

If the remote entity, however, is using security functions which are capable of interpretation by the local system after translation, then communication is possible providing the data is translated appropriately. The IDCA is responsible for requesting the appropriate mechanisms via the SSA to attempt the translation. Dealing with these kinds of incompatibilities is difficult, but is representative of the sort of problems that will be encountered when connecting diverse real networks.

5.3. Centralised Control

The most common reaction to the requirement for an organisation to implement security is to centralise control of all the information flow within the organisation in an attempt to gain complete control over the distribution of data and resources within the organisation. This is typified by military establishments where all confidential information is held in a central *registry*, and users requiring access to such data must go through the registry to obtain it. This is a powerful approach for small systems where the number of users and the quantity of data is such that one control centre can effectively manage the information database.

A centralised approach is useless, however, for large systems because the quantity of data to be stored and managed is too great. Access time to the database becomes prohibitive and effective management is impossible. Once control over the access to the data is lost or becomes ineffective, the whole purpose of the control is destroyed and the security is easily violated. The world telephone system can be used as a convenient analogy. It would clearly be impractical for the entire telephone network to use one exchange. Not only would access times render the system useless, but the effective control of the exchange would be impossible. Instead, the telephone network uses a distributed approach to management, whereby a large number of controlling exchanges are used, and a limited number of users are hosted on each exchange. This *clustering* technique is a powerful approach to reducing the size of the problem, and is the basic idea underlying the concept of the *Security Management Centre.*

5.4. Security Management Centres

As mentioned when discussing the conceptual model, the security processor for a large distributed system will not physically reside in one location, but will be itself distributed throughout the system. Indeed, such an arrangement may be advantageous. One approach is to adopt the concept of a *Security Management Centre* (SMC), which acts as a central security 'exchange' analogous to packet-switching centres in data networks and which will be responsible for the management and control of secure activities on the network. This will include duties such as third-party provision and verification of public keys, notarisation, registration and certification services and association policing to ensure the integrity of a secure communication between two users throughout the duration of that association. In section 2.3. the reasons for the *denial of service* problem having been largely ignored in the past were suggested. The introduction of the SMC as a *trusted* third party goes a long way towards solving this problem, because the association may be supervised dynamically to allow for changing *quality of service* throughout the duration of the association. In a large system with users hosted on a local SMC, the SMC authenticates the local CSS processor and the local CSS processor authenticates the user. The control of the system in this way by the SMCs develops a *trusted path* across the system from SMC to end user, thus ensuring that system misuse is minimised or eliminated with the advantage that peripheral hardware and software need not itself be trusted, see section 2.6. Should the SMC supervisory functions detect that the association has been disrupted beyond that attributable to poor service quality, then it may be reasonable to infer that a possible denial of service attack may be in progress. Under these conditions, the SMC will clear down the association in a controlled manner, conducting what amounts to a damage limitation exercise. At worst, therefore, the *availability* of the system will be compromised and *not* its integrity.

All operations within the network that involve security will be controlled by the local SMC(s) and direct association with them be made via the local CSS hardware and the OSI channel. Each SMC fully controls a number of user terminals hosted upon it, determining authentication, general user access rights and privileges which the SMC holds in its Security Management Information Base (SMIB). Secure communication across the network would involve:

1. protocols linking each end user terminal to the host SMC;
2. protocols linking SMC to SMC.

The system would provide full flexibility of services irrespective of the location of the user within the network.

Since the agents comprising the CSS need not physically reside in one location but may be distributed throughout the system, fully distributed implementations require additional encryption services to protect data between remote agents or parts of agents. In purely local systems where the entire CSS resides in a tamper-proof module, such additional protocols are not necessary. Figure 5.2. shows an example of the SMC concept.

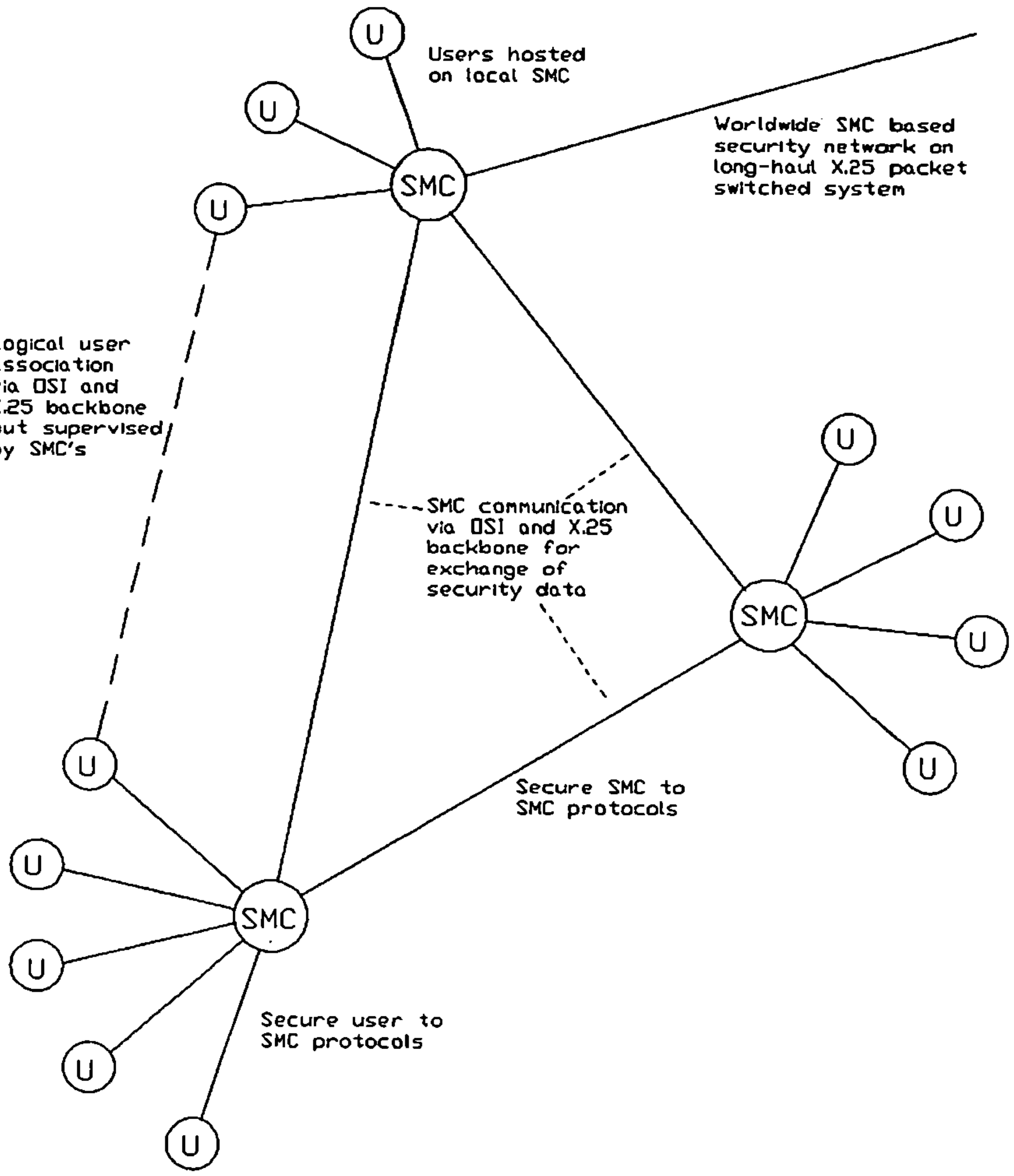

Users hosted on local SMC
Worldwide SMC based security network on long-haul X.25 packet switched system
SMC
U
Logical user association via OSI and X.25 backbone but supervised by SMC's
SMC communication via OSI and X.25 backbone for exchange of security data
Secure SMC to SMC protocols
Secure user to SMC protocols

Figure 5.2. The SMC Concept

5.5. The Protocols between Agents

The agents of the CSS described earlier communicate amongst each other by means of *protocols*. By their nature, these protocols must be as secure as the functions of the agents they facilitate communication between. One of the main problems associated with security models is the existence of the *covert channel*. A covert channel is a conceptual or logical path for information that can only be observed in the sense that it may be possible to duplicate their operation, but they cannot be explicitly monitored. The most promising way to overcome these difficulties lies with the use of cryptographic protocols. As described in section 5.4. the protocols are used to build up a trusted path between end users. The protocols of the CSS may be considered to form a three layer hierarchy:

1. **Global** protocols between SMCs;
2. **Local** protocols between the local SMC and the hosted users;
3. **Internal** protocols between agents of the CSS.

This distinction is purely logical and in practice all protocols are, of course, inter-agent protocols. The hierarchical approach, however, makes the description of the protocols easier in the same way that a high-level discussion of security services can be carried out without reference to the underlying mechanisms that facilitate their operation. The research described in this thesis covers (b) and (c) above. There are a number of protocols, but many are common to all agent interactions.

The fundamental protocol of the CSS is *authentication*. All transactions between agents require authentication before data exchange can commence because each agent must be assured that it is in fact communicating with a genuine user or peer and is not being deceived by an impostor. Many protocols for authentication have been described in the literature [SEBE,1988] and have been tried with varying degrees of success. The authentication protocol chosen for authentication between agents in the CSS and its formal analysis are discussed fully in Chapter 6.

Other protocols used by the CSS include those for key distribution, which allows end users confidently to share a secret session key and hence set up a secure association between themselves (supervised by the SMC). This protocol is also described along with a formal validation in Chapter 6.

Conceptually, the CSS protocols are generated in exactly the same way as the security mechanisms. Indeed, the provision of the underlying cryptographic functions is the main application of the security mechanisms. A confidence in the security of the cryptographic algorithms used, however, is no guarantee of the strength of the protocol. It is alarmingly easy to devise protocols based on very secure ciphers whose logic is faulty and which are all but useless in practice. While it may be possible to detect glaring errors by inspection, the more subtle loopholes will nearly always remain undetected unless a rigorous analytical tool can be applied to verify all the logical security aspects of the protocol. This is the purpose of the formal system described in Chapter 6.

5.6. Basic Implementation

Using the ideas developed in connection with the concepts, agents and protocols of a Comprehensive Security System, a practical system has been implemented using a Novell Ethernet LAN comprising three IBM Personal Computers configured as an SMC and two user nodes. This minimum configuration is sufficient to model and test:

1. the cryptographic algorithms;
2. the user to SMC protocols;
3. the inter-agent protocols;
4. the communications interface.

It does not provide for a demonstration of SMC to SMC protocols. It is envisaged that future research would include the connection of the LAN to a remote end system, possibly via a national WAN, so that these protocols could be tested as well. The basic communication paths within a local CSS processor are shown in Figure 5.3.

5.6.1. The CSS Hardware and Software

The CSS hardware for the demonstration system is quite simple but very effective. The purpose of the secure hardware is to force the terminal to boot into the CSS software system and prevent any attempts to disable this occurrence. The design of the PC firmware allows for the existence of extension ROM adapters which can initialise certain system operations during the *power on self test* (POST) sequence. This sequence is automatic and cannot be aborted and takes place before any disk drives are addressed or even recognised by the system. The normal use of the ROM adapters is to allow for the replacement of default system hardware such as video drivers by upgraded units to allow for system enhancement. The CSS hardware uses this facility for a novel purpose.

The unit comprises of an installable ROM adapter memory mapped into the 8086 address space at segment $D800 which is unused by the vast majority of PCs. The circuit diagram of the hardware and source code of the ROM firmware is given in Annex 1. Due to the format of the ROM firmware header, during the POST sequence the ROM is located and *execution is passed to it.* The ROM then carries out several actions, namely it:

1 issues a title banner proclaiming that the computer is under the control of the CSS;

2. disconnects all the floppy disk drives in software by redirecting the appropriate interrupt vectors and installing a hardware based interrupt handler which simply flags disk read errors and returns;

3. forces boot from fixed disk.

When the system boots from the fixed disk, the **CONFIG.SYS** is executed; again, this process cannot be aborted. The first command in the **CONFIG.SYS** file is to a *device driver* that requires a password to be entered before boot will continue. When the password is successfully entered, the interrupt vectors hooked by the ROM hardware are rechained, restoring the functionality of the floppy drives. In this way, the CSS hardware/software has *authenticated* the user and so a trusted path exists between the CSS and the user.

When user authentication is complete, the CSS kernel is executed, again via a device driver, and system startup continues to the familiar C:\> prompt. The installation of the local CSS software module cannot therefore be circumvented without physically opening the computer and removing the card.

5.6.2. The API for the PC System

As stated in the discussion of the conceptual model, the specific design of the Application Program Interface (API) is largely dependent on the host system. In the PC implementation, the host operating system is MS DOS and the API was designed accordingly. The API is complex and the complete software interface is included on the floppy diskette accompanying the thesis.

The CSS was written as an application layer *memory-resident* software module in Pascal and 8086 assembler with *interrupt vectors* controlling requests to operating system functions. In addition, the CSS monitors its own interrupt, which is used for passing security oriented requests to the CSS kernel. This distinction is necessary because the API of the CSS is designed so that the CSS can be invoked in one of three ways:

1. the *user* can press a *hot-key* combination to activate the CSS directly. In this mode, the user can request a number of services directly, such as authentication at the start of a secure session on the terminal, or direct encryption of a file he wishes to transfer to floppy disk and send through the post;

2. the *foreground application* can request security services of the CSS via a software interrupt reserved for the purpose. This would be used, for example, in part of a secure e-mail package which required the services of encryption mechanisms under the control of the CSS;

3. the CSS monitors all system requests for operating system services which may affect the security of the system. These include such services as file I/O, port control and execution privilege.

It is important to note that the first two modes are *voluntary*, that is, the security services are available purely on a request by request basis, with no requirement to actually make use of the services offered. The third mode is *compulsory* and cannot be circumvented by the system.

For example, in the demonstration system, all files have additional attributes specifying their security level as logical system *objects*. All file I/O requests are monitored by the CSS. Should an application, such as a text editor, attempt to open a file, the CSS intercepts the DOS request and determines whether the user has sufficient privilege for the operation. It can determine this in one of two ways. Firstly, if the application is purely local, access authorisation data to the object may be held in the local SMIB. If so, the CSS can check the capabilities of the subject user and grant access without reference to external authority. If the data is not held locally, however, the local CSS processor establishes contact with the SMC via the LAN, where the main SMIB is held. The CSS consults the SMIB which holds the rights and privileges of all the system *subjects*. If the capability of the *subject* is equal or superior to the classification of an *object* then the operation is permitted. If not, the operation is denied. The above description assumes that the user has *voluntarily* accessed the CSS at the start of the terminal session, and identified and authenticated himself to the system. If he has not done this, then on the first occurrence of identification/authentication being required, the CSS will suspend execution of the foreground application, and open a window requiring the user to identify/authenticate himself. Connection is then made with the SMC and SMIB as before, and authentication proceeds. If the user is accepted, the CSS closes the window and resumes execution of the foreground task. It is possible that the relevant data is not held in the SMIB of the local SMC. This may occur as a result of the user normally being hosted on another SMC and being a *guest* on the local system. In this case, the local SMC would request the necessary data from the remote SMC via an SMC to SMC protocol, although such protocols are not implemented on the demonstration system.

The PC CSS API is implemented as a series of *interrupt handlers* to monitor software interrupt activity. In this way, calls to the operating system can be monitored as well as providing a convenient, address independent way of calling functions directly from the CSS itself. These are:

1. INT $E0 which is an unallocated interrupt, and is used to pass voluntary requests from application processes to the CSS kernel for security services. The parameters associated with the request are built into a formalised data structure, and the service request code is placed in the AH register in accordance with 8086 software interrupt calling conventions. Software interrupt $E0 is then called, and the CSS, monitoring the interrupt, intercepts the request and acts accordingly;

2. INT $09 which is the keyboard hardware interrupt is monitored for the *hot-key* combination. If the user has pressed the hot-key to invoke the CSS voluntarily, a window is opened to receive user data;

3. INT $21 which is the MS DOS function despatcher. The CSS monitors the AH register on entry for operating system service requests of interest, and acts accordingly upon them. Requests which are not of interest are simply passed on to the operating system. The monitoring of this interrupt is the compulsory component of the CSS activity and in effect categorises this aspect of CSS activity as a *security kernel* standing between applications and the operating system.

By way of example, the code for one of the INT $21 handlers is shown overleaf with detailed comments. The source code for the CSS is large and the rest of the code may be found on the floppy disk bound within the master copy of this thesis.

```
(* Set up memory requirements for interrupt handlers *)
{$V-}
{$M 20000,0,0}
{$F+}
(* Watch for any operating system requests for the FILE OPEN function.  This function
is always the precursor for any file read/write operations under MS DOS *)
PROCEDURE Int21Open (VAR IntRegs : IntRegisters);
begin
WITH Intregs DO
begin
        (* get the filename to be opened from the DS:DX reg pair *)
        OFile := '';
        i := 0;
        REPEAT
                OFile := OFile + char (Mem [DS:DX+i]);
                Inc (i);
        UNTIL Mem [DS:DX+i] = $0;
end; { with Intregs do }
end;
(* This is the main Int $21 Handler which traps ALL DOS operating system IO requests.
The handler checks the operation for security-oriented implications, and calls CSS
services as appropriate *)
PROCEDURE Monitor (BP: word); INTERRUPT;
VAR
IntRegs : IntRegisters absolute BP;
begin
InterruptsOff;
```

```
(* This part of the handler looks for DOS function $3D which is file open. If the handler
is awake and the user has not already authenticated himself to the CSS then
authentication services are required and called *)
IF (IntRegs.AH = $3D) AND (Awake) AND (NOT AlreadyAuthenticated) THEN
begin
        OurIntStackPtr := @OurIntStack [SizeOf (OurIntStack)];
        SwapStackAndCall (@Int21Open,OurIntStackPtr,IntRegs);
end;
(* rechain to the old INT $21 vector after use *)
InterruptsOn;
ChainInt (IntRegs, ISR_Array [MonitorHandle].OrigAddr);
end;
{$F-}
```

5.6.3. Management of the SMIB data

In the demonstration system, the SMIB data is held at the SMC which is the network file server. We have chosen to require the system manager to be physically present at this machine when amending or updating data in the SMIB. It is common practice to keep the network file server in a physically secure and conducive environment, such as a locked, air-conditioned compartment. While it would be quite simple to allow a higher level of system privilege at one of the user workstations to allow the system manager to log on with system privileges and amend the SMIB data remotely, this was not done as the additional security afforded by the physical access constraints to the SMC is consistent with the high security status of the system.

5.6.4. Secure Communication across the LAN

The PC implementation encrypts data over the LAN using the Quadratic Residue Cipher (QRC). A detailed exposition on the operation of this cipher can be found in [SHEP,1990a] and a copy of this paper is bound into the thesis for convenience. Realistic security is used with large integer arithmetic being carried out by carefully optimised assembler routines. Even allowing for the fast software however, the cipher is slow, taking around ¼ second per character for 200 digit keys. Factoring such numbers is way beyond the capability of current technology (on the order of 15 million years) and can be considered quite safe for all practical purposes, see section 6.4. A hardware module is being developed which will implement the fundamental arithmetic operations in hardware, and will serve as a general-purpose 'cryptographic engine'. It is expected that use of this hardware and will considerably speed up the ciphering operations [SHEP,1990a].

The CSS communication software is written in a modular form to allow easy amendment for different network systems. The Ethernet card in a PC can be considered as providing the OSI stack up to layer 3 but with some of the transport functions of layer 4 as well. The CSS itself is an application entity and its only interface with OSI is with the presentation layer. Clearly, in a completely homogeneous system such as a PC LAN, there will be no syntactical differences in data representation across platforms and so *normal data transfer* takes place across the presentation layer (see section 3.3.3.). Thus, in order to preserve the integrity of the OSI model, the CSS contains null presentation and session layer modules and in fact passes data via these to the transport layer interface of the Ethernet card. The internal data communications of the local CSS processor are shown in Figure 5.3.

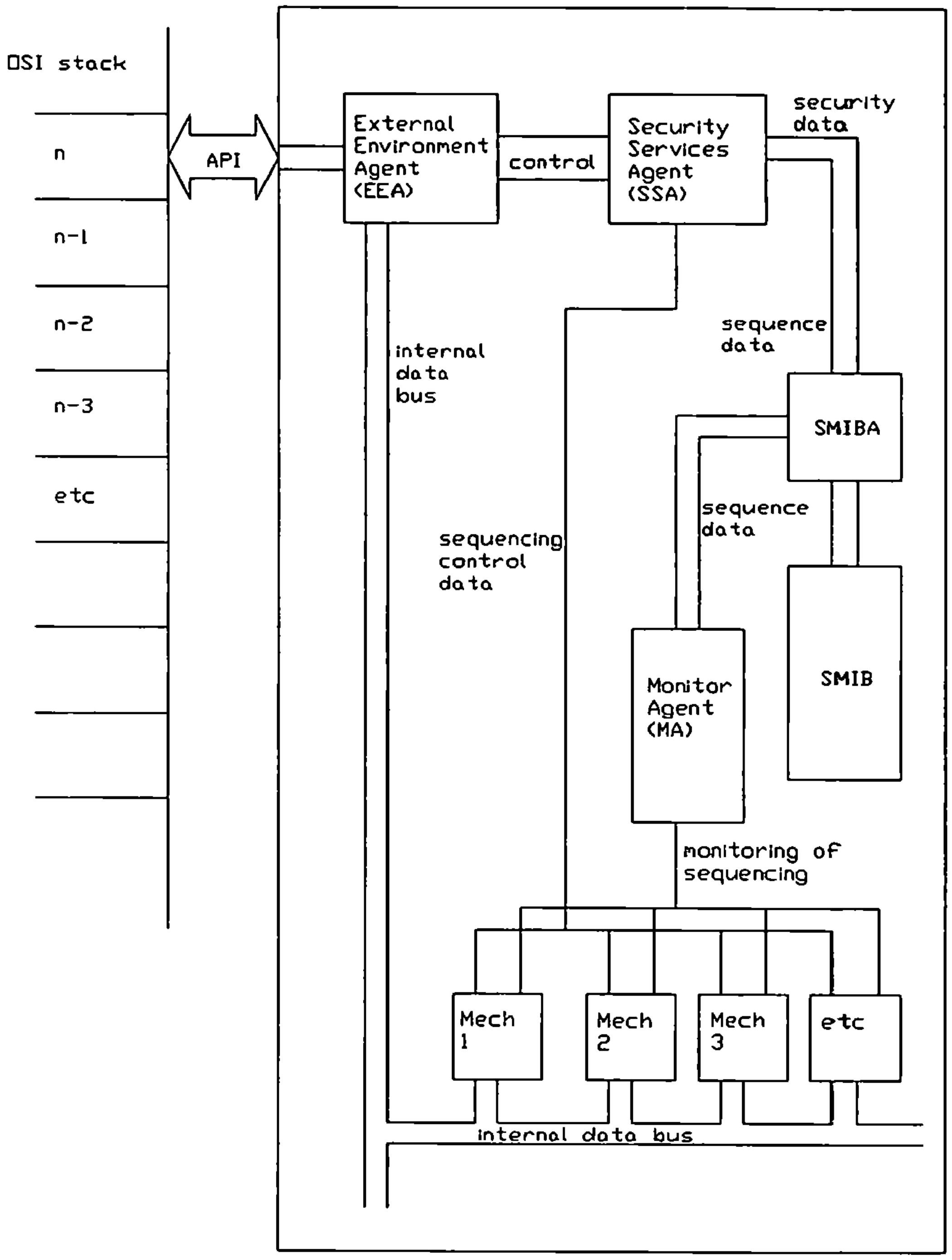


Figure 5.3. Agent Communication within a Local CSS Processor

5.6.5. Other Security Measures

The system also implements the following features:

1. timeout after a pre-determined period will clear down the connection with the SMC, and place the CSS kernel in a dormant mode. This avoids the possibility of a user logging onto a terminal, making a successful association with the SMC, and then inadvertently leaving the terminal unattended, allowing an unauthorised person to carry out operation for which he is not privileged;

2. an audit file of system activity is maintained at the SMC for system integrity auditing purposes;

3. the association is continually *supervised* by the SMC. A brief research program into the identification of users from their individual typing characteristics was carried out. Although an analysis of a short fragment is inconclusive, it is possible to make a continuous assessment of the style of typing, and hence possibly identify a change of user part way through a session. The preliminary results of this research are detailed in [SHEP,1990b].

5.6.6. Limitations on the PC Demonstration System

The system as currently implemented suffers from the following shortcomings:

1. the agents functions in the CSS software were written for procedural convenience and are not completely functionally divided into separate modules as recommended in the conceptual model. This makes changing the function of a particular agent more difficult than it would be in a totally modularised implementation, but allow a substantial reuse of code and hence a less memory-hungry system. Memory requirements of resident software are very important with the limited memory available in a PC and the size of modern foreground applications. In a realistic system, presumably written and maintained by a large number of programmers, it is very important to modularise the software as described to allow independent function development and maintenance;

2. the encryption/decryption times would need to be significantly improved, as sending a large file with the present system would be prohibitively slow;

3. in a completely secure system, it would be necessary to monitor and control other interrupts as well. For example, the system timeout is dependent on the data read from the real-time system clock. To circumvent the timeout function, a user could keep resetting the clock every few minutes, to deceive the CSS into believing that no time had passed. It would be necessary to monitor and control interrupt $1A, which controls the setting and reading of the real-time clock. The CSS would then make resetting the clock a privileged operation. (The notion of *timeliness* and the systems *beliefs* about time are critical for the analysis described in Chapter 6);

4. the demonstration system comprises only of a single SMC and two users. The system cannot therefore demonstrate and validate inter-SMC protocols.

6. A NEW ANALYSIS METHODOLOGY

6.1. The Need for a Problem Reduction Technique

As discussed in section 4.8. the major difficulty with any analysis of a large system with many parameters and variables is the scale of the problem. To define precisely the system state at any time in terms of every possible variable presents problems of combinatoric complexity which are impossible to overcome. In any case, analysis of this sort is not very useful for the CSS application. By way of a loose analogy, it is possible (in theory) to define the thermodynamic state of a closed system by recording the position and momentum of every particle, but it is far easier and more useful just to record the temperature!

The security analysis methodology described in this thesis, therefore, is based fundamentally on the ability to reduce a very complex sequence of operations to a much simpler set of parameters which are capable of measurement and analysis.

The basic design principle which will admit the sort of analysis described is based on the decomposition of the entire CSS software into a few very small units. All the security aspects of the CSS will be capable of being built up from these units, albeit large numbers of them. Each unit is based on very simple mathematical functions which are combined to yield other functions with secure properties, *trapdoor functions*. Each of these functions is rigorously analysed in terms of security, based upon the *complexity* underlying certain problems in number theory currently believed to be intractable, When each is validated, it is placed in a conceptual 'library' where it can be called when required. Any functions built out of the validated units will themselves be secure, and thus a layered approach to the construction of the security functionality can be taken.

The security functionality comprises two parts. The validated units are combined to produce:

1. **Data Manipulation Mechanisms (DMM)** such as encryption algorithms to allow security transformations to take place on the data;

2. **Protocol Generation Mechanisms (PGM)** such as authentication protocols to allow the agents of the CSS to communicate with each other in a secure manner.

The analysis methodology to be described therefore also comprises of two parts:

1. a **recursive logic** is used to *monitor* and *supervise* the sequencing of the validated DMM and PGM units which generate the security functionality;

2. a **modified Hoare logic** is used to validate the security of the *protocols* generated as a result of the operations described in (1).

Each aspect of the CSS security functionality is firstly decomposed into functional blocks. This is carried out for each security function the CSS will be called upon to perform. Many of the blocks will ~~to~~ be common to most, if not all, of the functions and hence a very efficient and compact system results. When the blocks are determined, they are placed in a secure function 'library'. This may be a true software library in the case of a pure software implementation or may comprise of physical hardware mechanisms in the case of a hardware implementation. To add a new function to the CSS, it is merely necessary to decompose the function and add the new blocks to the library, although in practice all the necessary blocks may already be present. To implement a function, the blocks are sequenced in the correct order. In this way, both mechanisms and protocols can be generated. The sequencing is very important to the security of the CSS and so the sequencing data is held in the SMIB. The approach to the design and analysis is illustrated schematically in Figure 6.1.

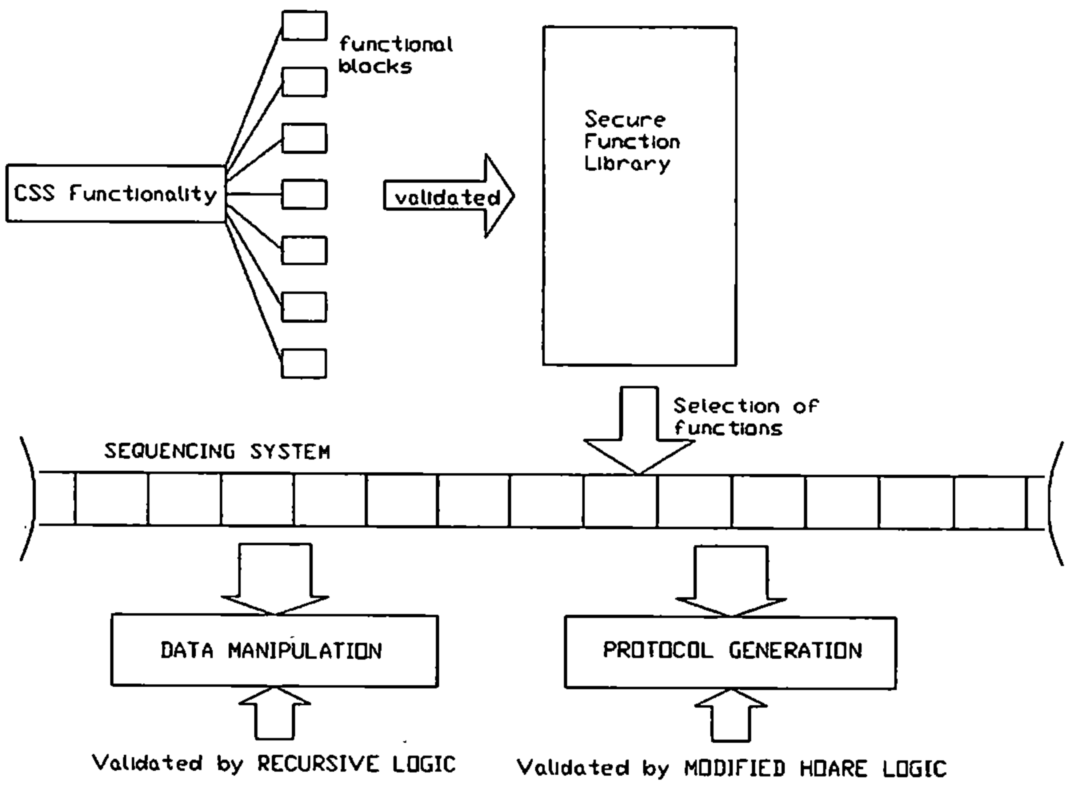


Figure 6.1. Approach to the Design and Analysis of the CSS

6.2. A Problem Decomposition Methodology

The problem decomposition methodology used for the CSS relies on breaking down the security functionality into layers of state machines. Each layer comprises of *nodes* which are the *roots* of *trees* in the layers beneath. The correct enforcement and monitoring of the sequencing of these machines is critically important to the analysis methodology. Unless the rules of *n*-ary tree structures are rigidly observed, the analysis breaks down. From these simple trees, all the mathematical functions and software control structures required by the CSS can be derived, and hence the analysis can be applied not only to the semantic correctness but also to the functional correctness of the system. These structures, as they apply to the analysis methodology, are now introduced.

6.2.1. Binary Tree Functional Decomposition

Consider a general abstract function that can be functionally separated into two parts, $f(x) = g(x).h(x)$. This may be represented by the *binary tree* shown in Figure 6.2.

It may be that one of the component functions, say $h(x)$ can itself be logically broken down into two functions, say $i(x),j(x)$. The tree would then take the form shown in Figure 6.3. The vertex of the tree is conventionally called the *root* and the end nodes are called the *leaves*. Note that a tree can only branch *out*, and can never branch 'back together'. Figure 6.4. shows some arrangements which are *not* trees. In a tree, every subordinate node is drawn below its superior, that is, there are no horizontal lines in the diagrams. A *branch* is described as *entering* a node if it comes from above. It is described as *leaving* if it goes to a subordinate node. If a branch leaves node **A** and enters node **B**, node **B** is referred to as the *offspring* of **A**. Node **A** is the *parent* of **B**. A node of one tree may be regarded as the root of another tree - a tree within a tree. The tree of which an intermediate node is the root is called a *subtree*, see Figure 6.5.

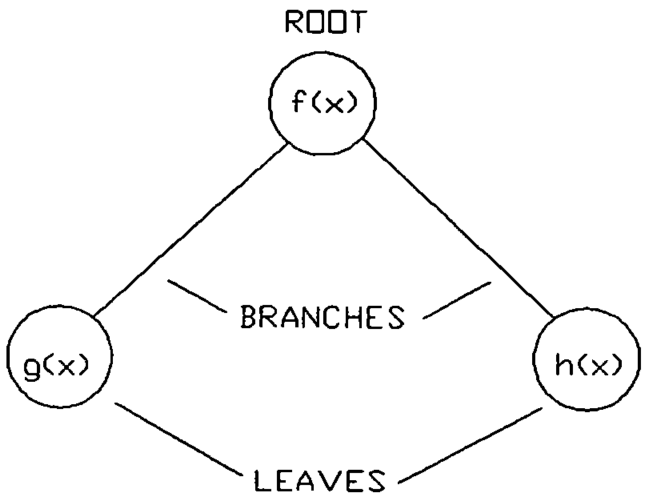


Figure 6.2. A Simple Binary Tree with Two Leaves

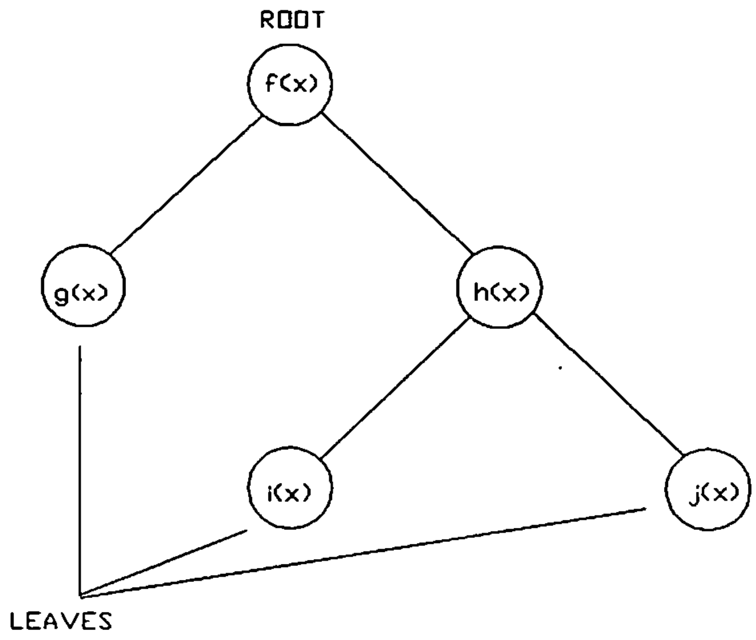


Figure 6.3. A Simple Binary Tree with Three Leaves

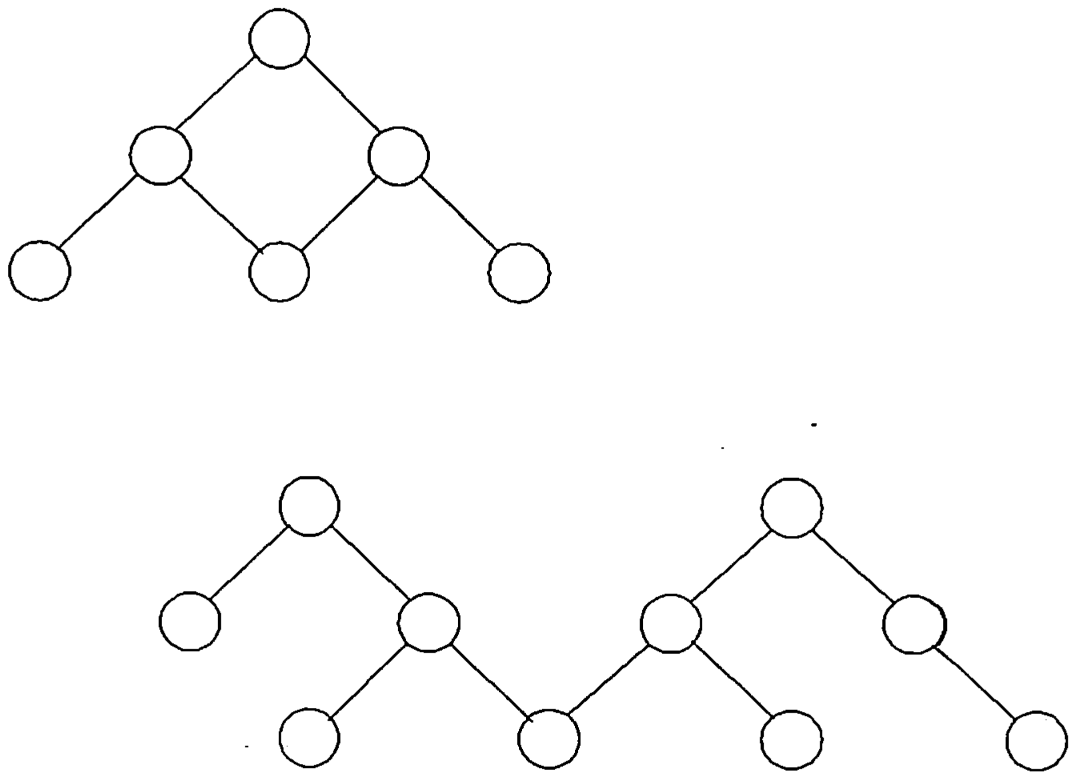


Figure 6.4. Binary Combinations which are *not* Trees

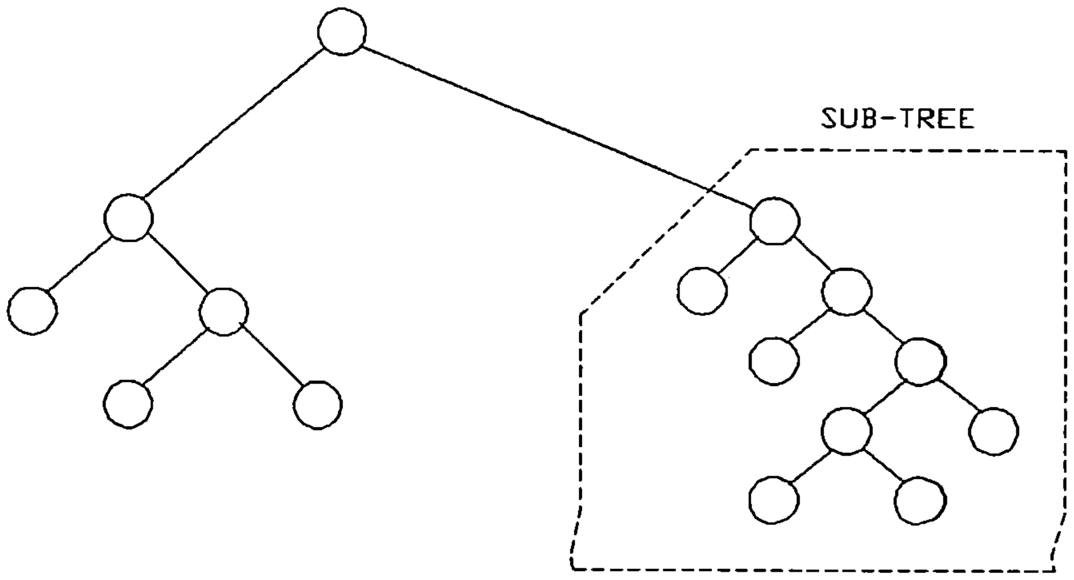


Figure 6.5. A Tree Containing a Sub-Tree

6.2.2. *n*-ary Tree Functional Decomposition

There is no reason why, in general, a tree should be restricted to only two branches per node. It is possible to have trees where each node has *n* branches, or even the most general tree where each node can have an arbitrary number of branches, see Figure 6.6.

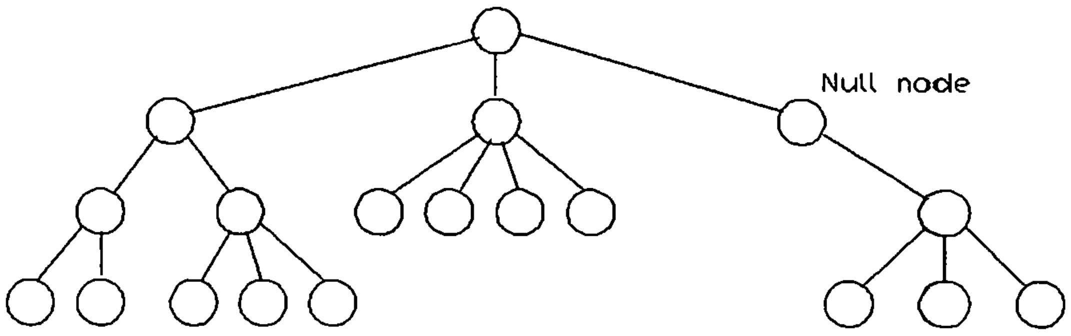


Figure 6.6. A General Tree with Arbitrary Numbers of Branches

Note that it is possible for the tree not to actually 'branch out' at a node, but simply have one branch entering and one branch leaving. In this research, such a node is designated a *null node*. It is upon the structure and properties of this general tree that the CSS design is based.

Each node in the general tree can represent a *function* of arbitrary complexity. A function has one or more *arguments* as its input and one or more *returns* as its output. In keeping with standard mathematical notation, the input argument or arguments are written on the right hand side of the function; the output result or results are written on the left of the function, for example:

y	=	SQUARE ROOT OF	x
return		function	argument

This idea, as applied to trees, is illustrated in Figure 6.7.

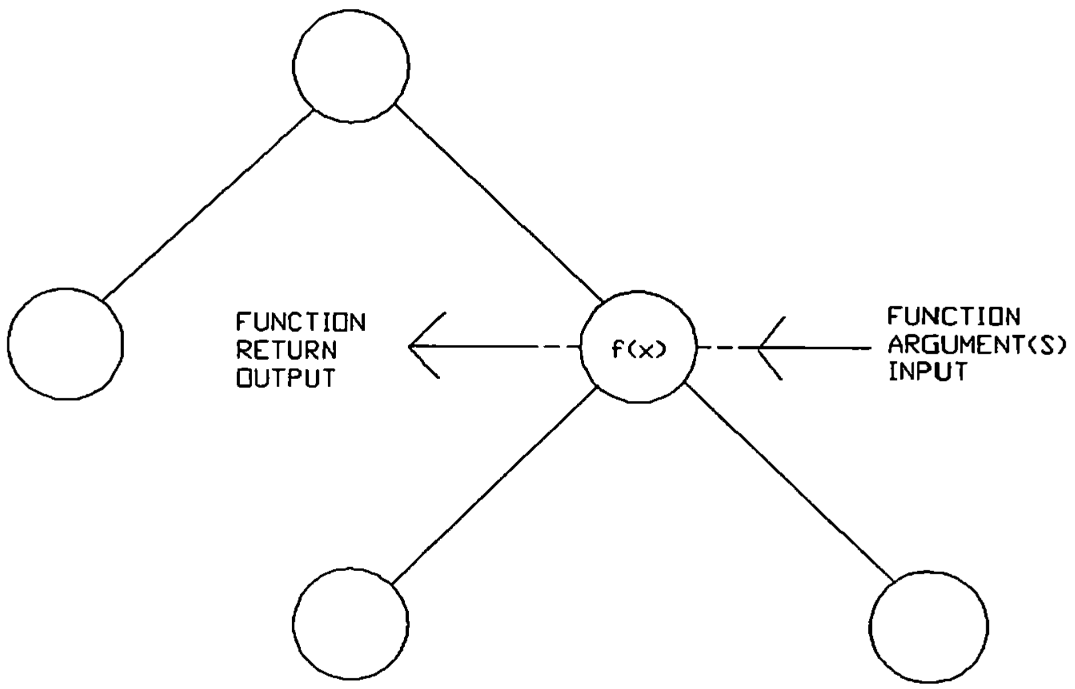


Figure 6.7. Conventions for Arguments and Returns of Functions

The importance of the notation is its generality. The function may be a programmed algorithm, a statement in a non-procedural language, a program or subroutine specification, or a broad general set of requirements such as

$$\text{CURRY} \Leftarrow \text{PREPARE-MEAL} \Leftarrow \text{(Ingredients, Utensils)}$$

Using general trees, the complex sequences of data transformations that comprise the functionality of the CSS are broken down into simpler functions. This process is repeated until the functions cannot be simplified further. At this stage we say they are *atomic*, and the very basic units are called *atoms*, see section 6.3.1. Once the functionality has been totally decomposed into atoms, the design can be implemented from the *bottom-up*. Recalling that the atoms are combined only in ways that can be shown to satisfy security constraints, a layered set of secure functions can be constructed that will carry out the required functionality in a rigorous manner.

6.2.3. Control Structures and Dynamic Graph Representation

The decomposition of a function into sub-functions (or offspring) is achieved in a rigorous manner by means of *control structures* [MART,1982]. Three structures are defined, called *join*, *include* and *or*. Each of these serves a different purpose, depending on how the function lends itself to decomposition, and are illustrated in Figure 6.8. together with the *dynamic graph representation* of the structure. The graph is a projection of the tree representation, and may be annotated with timing information and so forth. While the tree diagram shows how the function is decomposed, the graph shows the order and chronology of the decomposition.

JOIN

Let *f(x)* be a function that requires two operations that must be performed *sequentially* on the argument, and that the *order* is important. This is commonly found in such cases as:

$$y = (x + 3)^2$$

It is the brackets which denote the order of the two operations. The addition must be performed first, then the squaring performed on the resulting sum. If the squaring is done first, the result is always $x + 9$.

INCLUDE

In this case, let *f(x,y)* be a process with two arguments and two returns, but which are independent of each other. This may involve adding a quantity to one and squaring the other. Dynamically, this kind of function cannot be performed sequentially, because both operations would be performed on both arguments. Conceptually, the operations happen independently and in parallel, and in practice could be implemented on separate processors in a parallel architecture machine.

OR

This final control structure takes account of functions where the operation to be performed is conditional upon the value of one of the arguments.

The three simple control structures considered so far can be combined into four somewhat more complex control structures, which offer greater flexibility in the decomposition process. The additional structures, again following the notation of [MART,1982], are called CO-JOIN, CO-INCLUDE, CO-OR and CONCUR. The architecture of these four structures is shown both in tree form and in dynamic graph form in Figure 6.9.

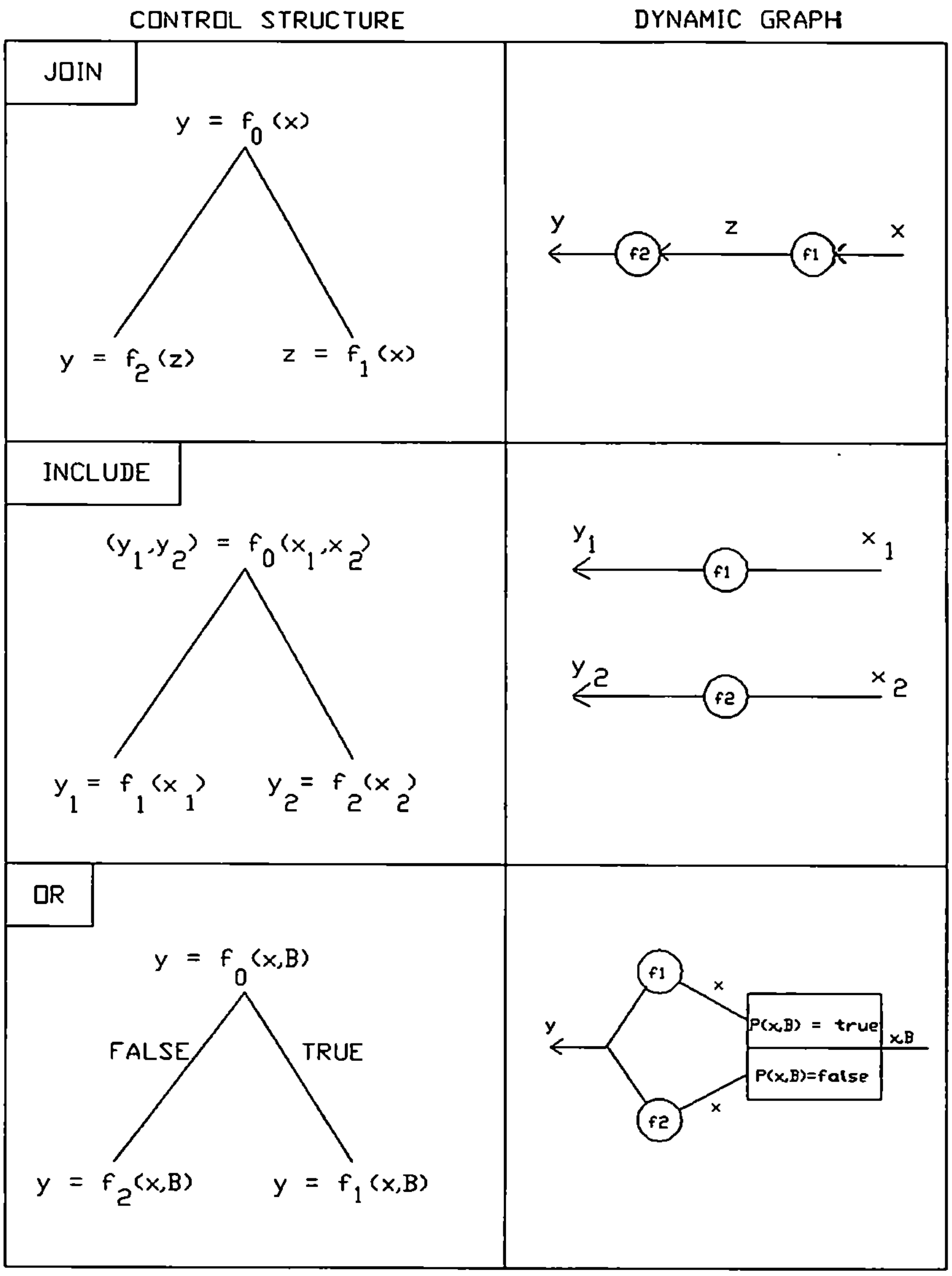


Figure 6.8. The *Join*, *Include* and *Or* Control Structures

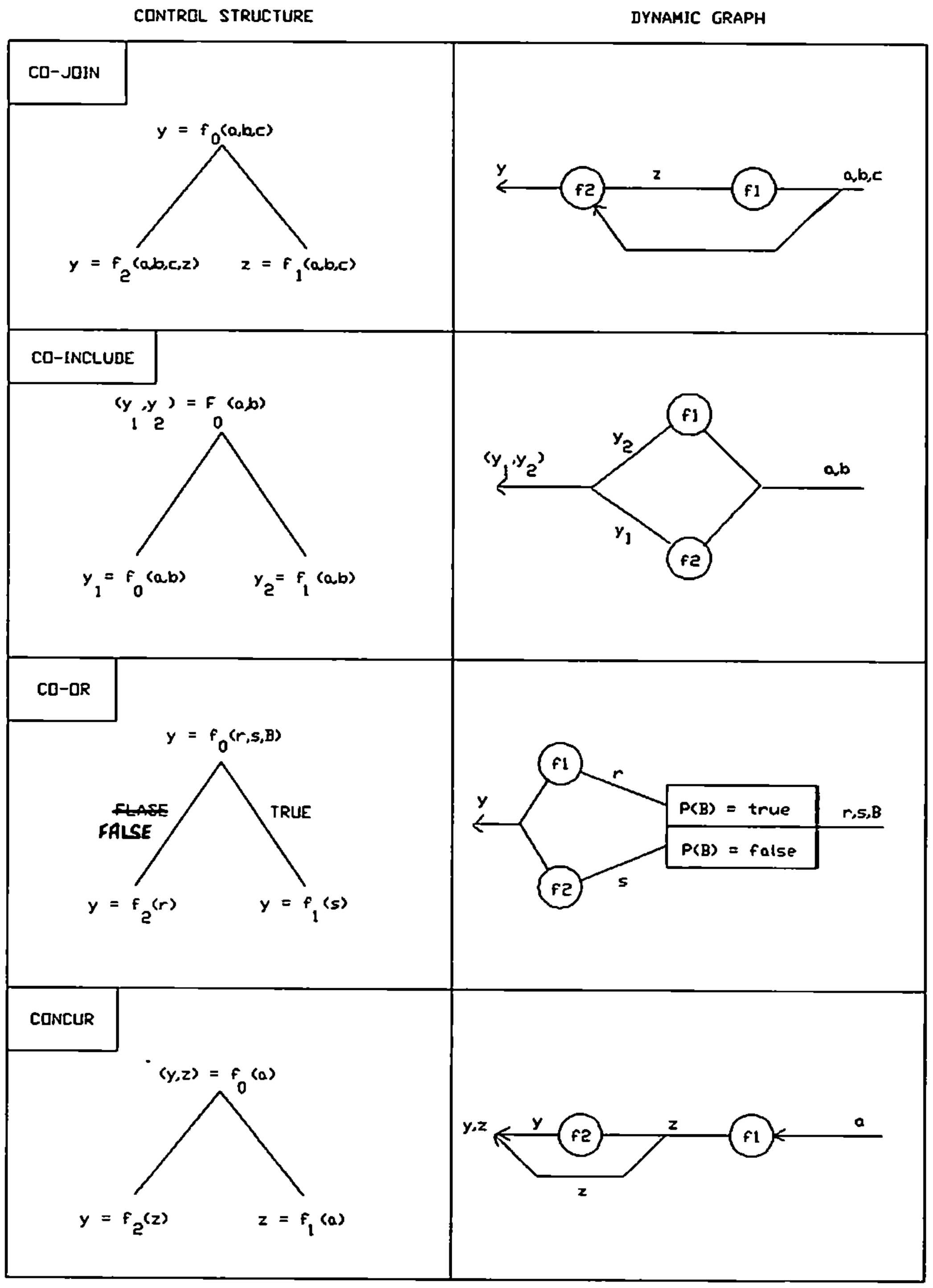


Figure 6.9. The *Co-Join*, *Co-Include*, *Co-Or* and *Concur* Control Structures

6.2.4. Recursion and Loops

Consider the evaluation of a factorial, written $n!$.

$$n! = n\ (n - 1)(n - 2).....1, \quad \text{where } n > 0.$$

We can use the following recursion for the calculation:

$$\text{Factorial } (n) = n. \text{ Factorial } (n\text{-}1)$$

The function $y = n$. Factorial (n-1) can be expressed in the tree diagram notation as shown in Figure 6.10a. To create the recursion, however, we require a stopping condition, that is, when $n = 0$. (If $n = 0$ then $0! = 1$ by definition). An **OR** expression may be used as the control structure to stop the recursion, see Figure 6.10b. Combining the simple function with the control structure yields the complete decomposition of the factorial function as shown in Figure 6.11. In this way, software constructs which are recursive can be expressed in the notation, and the element of control required for rigorous analysis of their execution can be expressed through combinations of the seven control structures described in the previous section.

In addition to recursion, programs commonly contain two types of *loop*. These are the **DO WHILE** and **REPEAT UNTIL** constructs. A **DO WHILE** loop tests whether a particular condition is true at the *start* of the loop, whereas a **REPEAT UNTIL** loop tests whether a certain condition is true at the *end* of the loop. In many applications, a particular requirement can be programmed using either construct, but the important difference between the two is that the **REPEAT UNTIL** loop will *always* be executed at least once, because the condition for looping is not tested until exit from the loop.

Some applications require the possibility of executing the loop *zero times*, and in these cases the **DO WHILE** construct is the only possibility open.

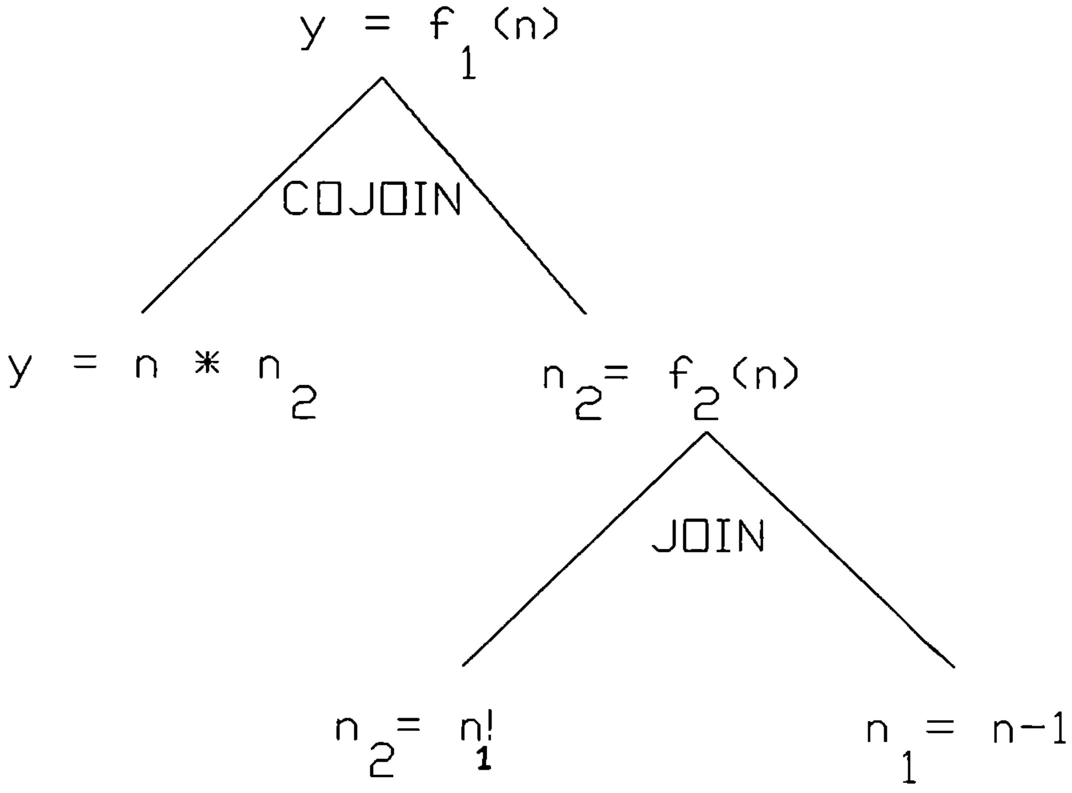


Figure 6.10a. Tree Representation for the Factorial Function

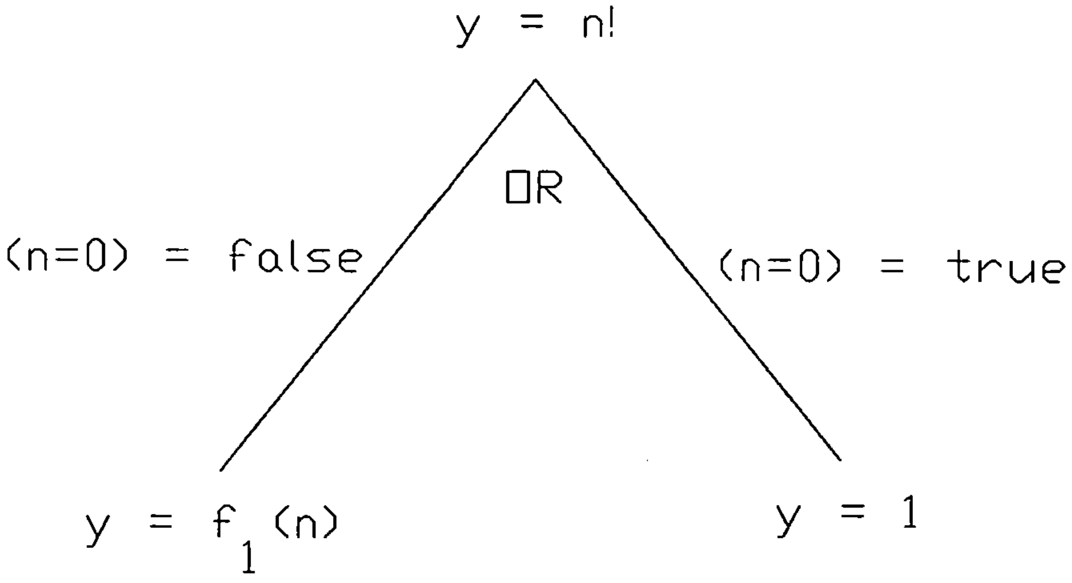


Figure 6.10b. Stopping the Factorial Recursion

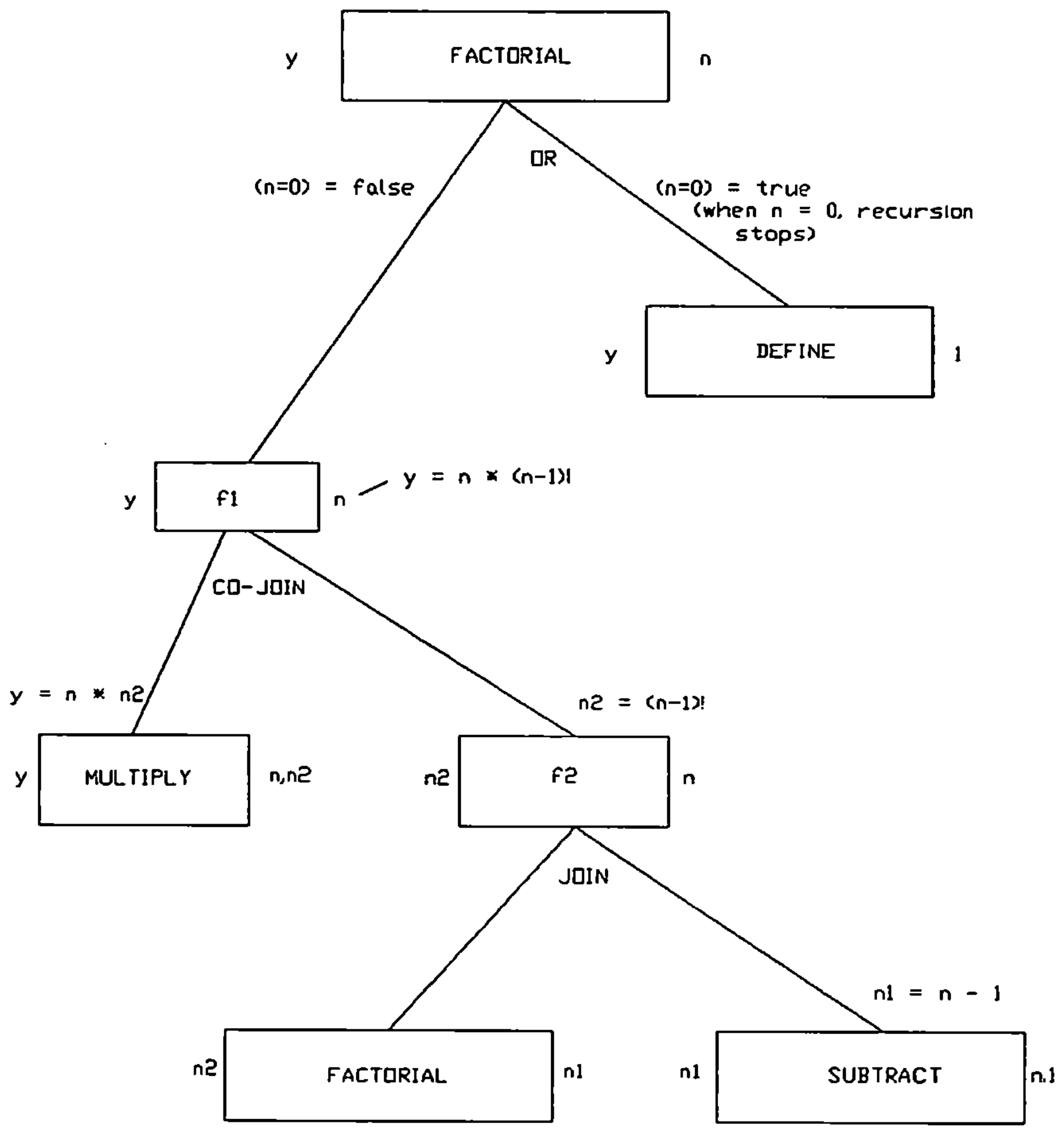


Figure 6.11. The Complete Factorial Function with Control Structure

6.3. The CSS State-Machine Sequence Reduction Logic

The reduction methodology described in the previous section can now be applied to the analysis of the CSS, using a hierarchical set of sequence definitions at a number of layers. The topmost layer within the CSS is the security *service* to be performed, while the lowest layer is comprised of the *atoms*, which are a small number of fundamental mathematical operations which can be performed on data, supplemented with a hardware real-time clock and any other non-mathematical hardware functions such as DES as required. Between these extremes, a number of other layers are defined. The complete hierarchical structure is defined as follows,

LAYER 1	SERVICES
LAYER 2	MECHANISMS
LAYER 3	COMPONENTS
LAYER 4	ATOMS

(Conceptually, a *layer 0* could be included to represent the overall security management function. This is embodied in the security policy from which the need for security services are determined.) The central idea behind the reduction technique is that of a *bottom-up* approach. By starting at the lowest layer, the *atom* layer, with a small number of provably secure data operations, the proof of security can be abstracted at higher and higher levels until the security service itself is shown to be secure.

Outside the physical boundaries of the CSS itself, the analysis can be extended from the security services to the security *policy*, and that policy provably analysed. The layers within the CSS are described in turn, starting with the lowest layer.

6.3.1. The Atom Layer

Ultimately, every operation performed by the CSS can be constructed entirely from a small number of basic *atoms*. These are:

1. a real-time clock;

2. an implementation of five basic mathematical operations:
   - (a) Addition (+);
   - (b) Subtraction (-);
   - (c) Multiplication (*);
   - (d) Integer Division to leave the remainder (MOD);
   - (e) Integer Division to leave the quotient (DIV);

3. a number of non-mathematical functions such as the Data Encryption Standard (DES), the Fast Encryption Algorithm (FEAL) and so forth if required, implemented in either software or hardware.

The five basic mathematical operations are combined to form the *modular exponentiation* (MOD-EXP) function

$$f(a,m,x) = a^x \pmod{m}$$

This function may be efficiently implemented in software using the repeated squares method [KOBL,1987,p.22], or may be even more efficiently implemented in hardware by other methods [MORI,1989].

6.3.2. The Component Layer

The modular exponentiation operator (MOD_EXP), together with the five basic mathematical operators, hardware real-time clock and other hardware/software atoms, are used by the Security Mechanism Agent (SMA) to construct a number of *components*:

1. time stamp generation, using real-time clock;
2. prime number generation by *Rabin's algorithm* [KNUT,1981], using MOD-EXP and basic function primitives;
3. random number generation by x^2 *mod n* generator [BLUM,1986], using MOD-EXP;
4. Euclid's algorithm [RIES,1987], using basic function primitives;
5. Euclid's extended algorithm [KNUT,1981], using basic function primitives.

While this list does not exhaustively cover all the algorithms which may be constructed from the primitives, it contains all the necessary and sufficient algorithms required to implement the services performed by the prototype CSS.

6.3.3. The Mechanism Layer

These five components are combined by the Security Mechanism Agent to produce the security *mechanisms* required by the CSS, which are:

1. RSA [RIVE,1977] Key Generation, using prime number generation and Euclid's algorithm;

2. block encryption with RSA, using MOD-EXP;
3. block decryption with RSA, using MOD-EXP;
4. QRC [SHEP,1990a] Key Generation, using Prime Number Generation and Euclid's extended algorithm;
5. stream encryption with QRC, using MOD-EXP;
6. stream decryption with QRC, using MOD-EXP and Euclid's extended algorithm;
7. digital signature with RSA, using MOD-EXP;
8. hash (digest) [ISO,X.509] with X.509, using MOD-EXP;
9. encryption with symmetrical algorithms such as DES, FEAL, using the hardware/software atoms directly.

Again, this list is not exhaustive, but covers all the mechanisms used by the CSS.

6.3.4. The Service Layer

The SSA uses the mechanisms to provide the services required of the CSS. These include those services recommended in 7498-2 and include,

1. confidentiality;
2. authentication;
3. non-repudiation;
4. integrity;
5. access control;
6. denial of service.

Note that (6) is an exception. It is only possible to *detect* denial of service attacks; it is not possible to stop them.

6.4. The Fundamental Security Assumptions

The reference point for the logical analysis of the formal security proofs is the set of initial *beliefs* about the system to be analysed. Implicit in these beliefs are certain assumptions which are made concerning the strength of the encryption and other data transformation functions used by the security services. Because these (and every other mechanism within the CSS) are constructed from sequences of atoms, certain assumptions concerning the security of the atoms from which the entire security framework is ultimately constructed are also crucial to the security analysis methodology of the CSS. The analysis relies on a combination of physical security (which is, in turn, a function of the implementation), the accepted validity of algorithms such as DES which have been exhaustively tested, and most importantly, two central problems in number theory which are *currently believed* to be intractable. These are the *factoring* problem and the *discrete logarithm* problem, and both apply to the important MOD_EXP atom, which is the most important building block of most of the mechanisms. These two problems and other closely related ideas are discussed under the general subject of *trap-door functions* [YAO,1982]. A trapdoor function is one where the computation of the function is trivial, but computation of the inverse function is intractable without knowledge of a secret parameter, the so-called *trapdoor*. In the case of MOD-EXP, it is trivial to calculate $y = f(a,m,x) = a^x \pmod{m}$ but intractable for large arguments to calculate the inverse. It is important to gain an idea of the theoretical time complexity estimates for the running times of algorithms which may be constructed to pose an attack on the security of the CSS.

It must be emphasised that, in common with all security systems using public key ciphers and other devices relying on these number theoretic problems, the absolute security of the encryption can never be *guaranteed*, because while no-one has yet published a polynomial-time factoring algorithm, no-one has succeeded in proving that no such algorithm exists. If such an algorithm were ever discovered, one of the fundamental beliefs about the system would be shown to be false, and therefore any conclusion drawn about the system security may also be false. It is important to note that this in no way undermines the analysis methodology, the main aim of which is the extension and abstraction of notions concerning beliefs about the *initial* system state to draw valid conclusions about the *final* system state, see section 6.9.

Individually, the dependence of the atoms, components and mechanisms on these principles is:

1. a real-time clock circuit encapsulated within a tamper-proof hardware module;

2. the RSA cipher relying upon the intractability of both the factoring problem and the discrete logarithm problem;

3. the QRC cipher relying only upon the intractability of the factoring problem [BLUM,1989];

4. the X.509 digest relying upon the difficulty of both the factoring problem and the discrete logarithm problem.

6.4.1. The Factoring Problem

The factoring problem defines the vast apparent difference between the amount of effort involved in multiplying some (prime) numbers together, compared with the effort involved in factoring the resulting product back into its prime factors. There are many factoring algorithms known which range from very simple but highly inefficient methods such as trial division through to highly sophisticated, special-purpose algorithms which rely on deep concepts in higher algebraic number theory.

The fastest general-purpose factoring algorithm so far devised is the *multiple polynomial quadratic sieve* (mpqs), which holds the record for the largest general factorisations so far carried out using a Cray XMP at the Sandia National Laboratories [DAVI,1985]. The expected heuristic running time of this algorithm is [POLL,1989]

$$O\ \{exp\ [c\sqrt{(ln\ \mathrm{N}\ lnln\ \mathrm{N})}]\}$$

where a bound on the value of *c* is not known. Given the fact that a 75 digit number can be factorised in about 10 hours with this algorithm (on the Cray), it is reasonable to conjecture that the factoring time for a 100 digit number would be around 2 months, and that for a 200 digit number to be around *15 million years*!

Certain special purpose algorithms have been devised which can factor numbers of special form very much more rapidly than the general purpose algorithms. These include Lenstra's *Elliptic Curve* method (ecm) [LENS,1986] which exploits the very rich group structure of elliptic curves, and Pollard's *Number Field Sieve* (nfs) [POLL,1989] which uses complex ideas of algebraic integers over suitably extended fields whose degree depends on the structure of the integer to be factored. The expected heuristic running time for the elliptic curve algorithm is exactly the same as that for the *mpqs*. Indeed, theoretical consideration of the distribution of primes and corollaries of the prime number theorem had led to the widely held belief that it was not possible to improve on running times of this order [POLL,1989].

The significance of the number field sieve, which is suitable at present only for numbers of the form $r^e \pm s$ for r and s sufficiently small, is that Pollard has shown it to have an expected heuristic running time of

$$O\ \{exp\ [c + \epsilon(ln\ \mathrm{N})^{1/3}(lnln\ \mathrm{N})^{2/3}]\}$$

where the value of c is around 1.526. This is considerably faster than hitherto thought possible, and effectively disproves the previously believed lower bound. Work is currently underway to try to extend the *nfs* to general integers [ADLE,1991]. If this proves possible, and the running time remains of the same order, then a significant advance will have been made in this area. The running time is, however, still exponential, and could be defeated simply by increasing the key size. The amount of extra work involved in using the larger keys grows only polynomially, but the factoring effort grows exponentially. It is thus always possible to 'outrun' an exponential algorithm by increasing the size of the problem instance.

It is clear that with the current knowledge of number theory, the factoring problem may be regarded as intractable for sufficiently large N, and hence that the belief in the security of mechanisms based on this problem is sound.

6.4.2. The Discrete Logarithm problem.

A logarithm is generally defined as the power to which it is necessary to raise a base in order to obtain a specified quantity. Within the field of real numbers, it is not appreciably more difficult to evaluate $x = \log_a y$ than to evaluate $y = a^x$. The discrete logarithm extends this idea to finite (discrete) fields, and the discrete logarithm problem defines the vast apparent difference between the amount of effort involved in raising a number to a power in a finite field as compared with the effort involved in determining the power given the discrete logarithm. That is, it is easy to calculate

$$y = a^x \pmod{q}$$

given a,x and q (for prime q and a a primitive element of the field), but very difficult, given y,a and q to evaluate x.

If $q = p^n$ is an odd prime power which is k bits long, experience suggests that the order of magnitude of time required to solve an instance of the discrete logarithm problem in $\mathbf{F}^*_q$ is comparable to that required to factor a k-bit integer [KOBL,1987]. That is, from an empirical point of view, the discrete logarithm problem seems to be about as hard as the factoring problem (although no-one has been able to prove a theorem to this effect).

It appears, therefore, that the discrete logarithm problem, for sufficiently large integers, may be considered a secure device, and again, any beliefs concerning the security of mechanisms based on this problem are sound.

6.5. Hierarchical Sequence Structure

Essentially, every security service can be broken down into a long sequence of simple operations on data variables. As described, these sequences are divided into different layers. The sequences defined within each layer may be regarded as finite state machines, which may call finite state machines in the layer below (as *procedures* or *functions*, with or without arguments), and be called by finite state machines in the layer above (again, as procedures or functions, with or without arguments). As long as the sequences defining the security services are carried out properly (that is, in the correct sequence and within specified time constraints) then the CSS can be shown to securely implement the desired security service.

The sequencing data for each of the services capable of being carried out by the CSS is stored within the Security Management Information Base (SMIB) of the CSS. Upon receipt of a service request across the API, the External Environment Agent (EEA) will route the *control* data header to the SSA and the *process* data to the CSS internal data bus, see Figure 5.3. The SSA will consult the SMIB via the SMIBA to determine the sequence of operations which need to be called in order to achieve the desired service, if allowed. At the start of the service, the status registers of the state machines will be cleared. As each operation is called in turn and completed successfully, the completion status bit is set in the status register. The MA supervises the sequencing by comparing the SMIB sequencing data with the status registers to determine the progress of the service, and to verify its correctness at each stage, see Figure 6.12.

Some examples of the flow sequences used to generate various security mechanisms and protocols are given in sections 6.7. and 6.8. respectively.

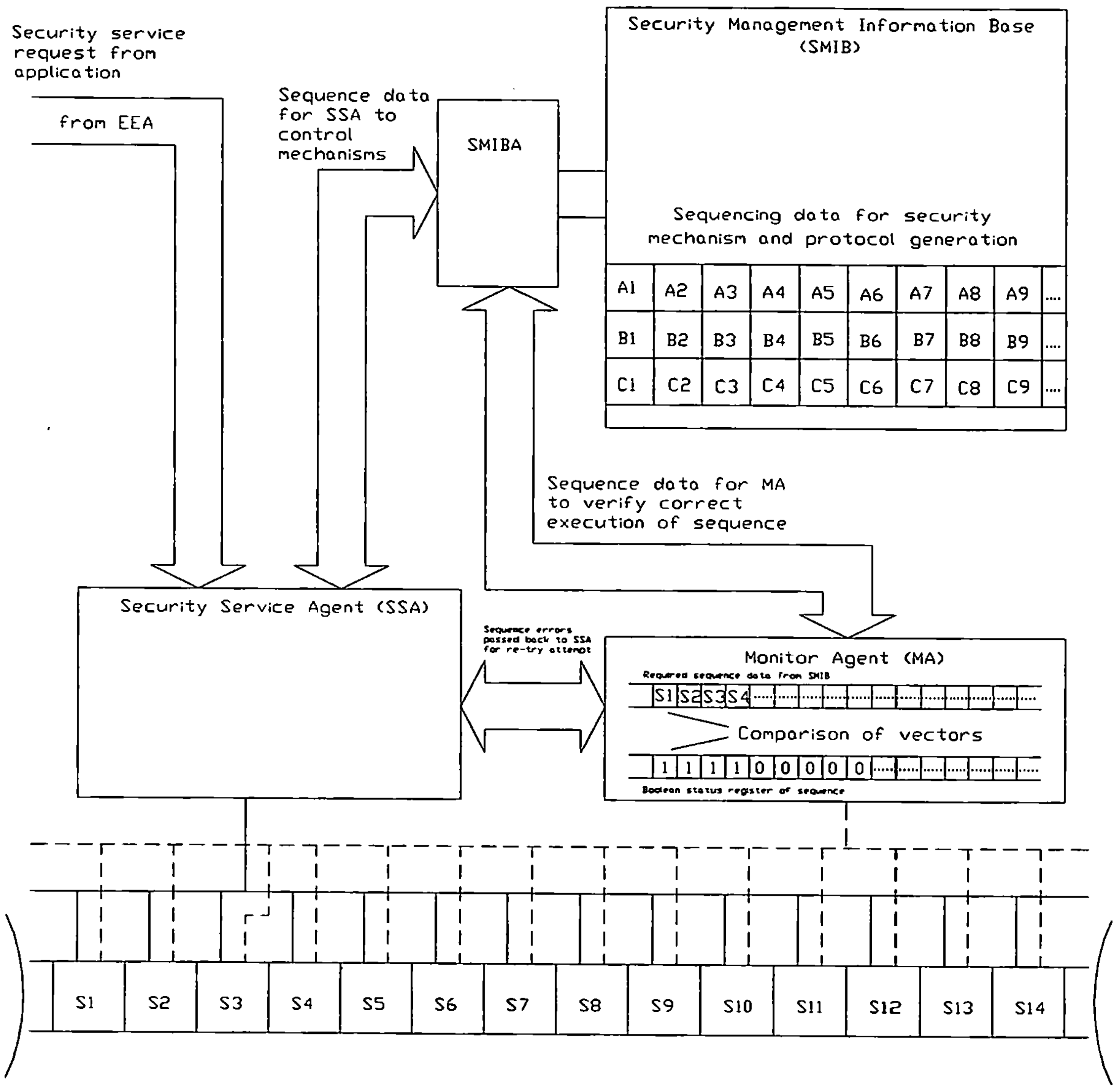

Security service request from application
from EEA
Sequence data for SSA to control mechanisms
SMIBA
Security Management Information Base (SMIB)
Sequencing data for security mechanism and protocol generation
A1 A2 A3 A4 A5 A6 A7 A8 A9 ....
B1 B2 B3 B4 B5 B6 B7 B8 B9 ....
C1 C2 C3 C4 C5 C6 C7 C8 C9 ....
Sequence data for MA to verify correct execution of sequence
Security Service Agent (SSA)
Sequence errors passed back to SSA for re-try attempt
Monitor Agent (MA)
Required sequence data from SMIB
S1 S2 S3 S4
Comparison of vectors
1 1 1 1 0 0 0 0 0
Boolean status register of sequence
S1 S2 S3 S4 S5 S6 S7 S8 S9 S10 S11 S12 S13 S14

Figure 6.12. Verification of Sequence Execution Correctness

6.6. Notation

The analysis of this system is facilitated by means of a special notation, see Figure 6.13.

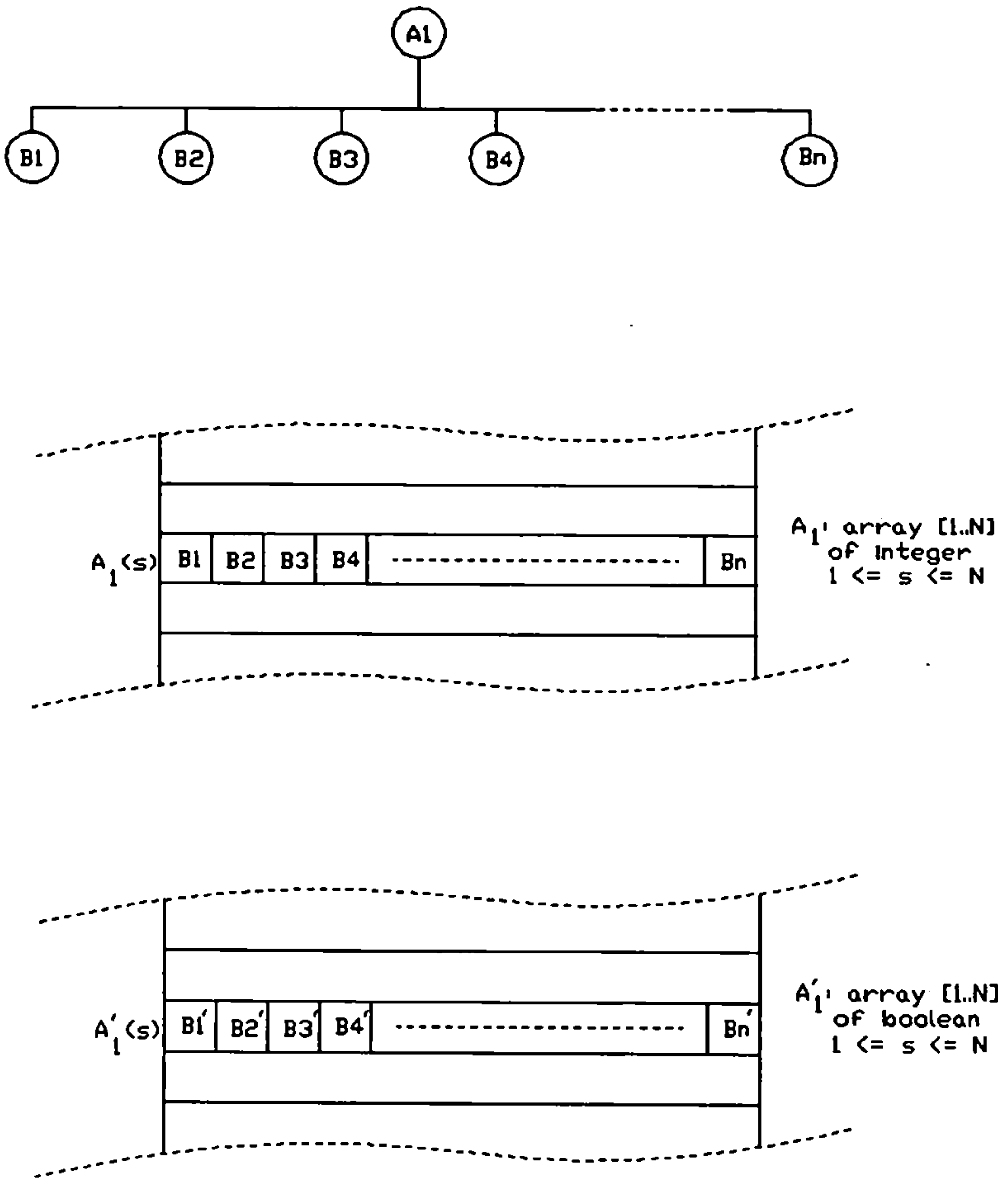

A1
B1
B2
B3
B4
Bn
A1(s)
B1 B2 B3 B4 Bn
A1: array [1..N] of integer 1 <= s <= N
A'1(s)
B1' B2' B3' B4' Bn'
A'1: array [1..N] of boolean 1 <= s <= N

Figure 6.13. A State Machine Representation within the SMIB

Let state-machine A_1 be represented by the sequence of state-machines $B_1..B_N$, let $A_1(s)$

represent the sequence vector in the SMIB containing the sequencing data of A_1, and let $A_1'(s)$ represent the boolean status vector containing the status data, $B'_1..B'_N$, of the success/failure of component state machines $B_1..B_N$ in A_1. Then the security state of state-machine A_1 after *m* steps can be determined from the recursive function $S(A_1,m)$,

```
FUNCTION S (A1, m : integer) : boolean;
begin
WHILE m > 0 DO
        IF ( (A1'[m]) AND ( S (A1,m-1) ) THEN
                S := true
        ELSE
                S := false;
end; □
```

6.6.1. An Example

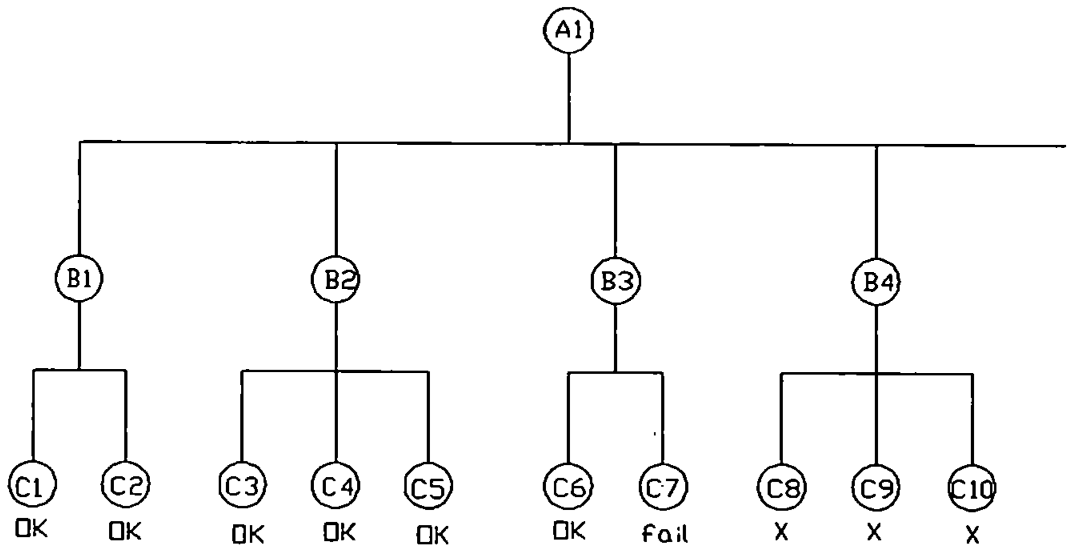


Figure 6.14. An Example of a Hierarchical Sequenced State Machine

Referring to Figure 6.14., suppose that

$\exists A_1 : \boldsymbol{A}_1[1] = B_1$

$\exists A_1 : \boldsymbol{A}_1[2] = B_2$

$\exists A_1 : \boldsymbol{A}_1[3] = B_3$

$\exists A_1 : \boldsymbol{A}_1[4] = B_4$

and

$\exists B_1 : \boldsymbol{B}_1[1] = C_1$

$\exists B_1 : \boldsymbol{B}_1[2] = C_2$

and

$\exists B_2 : \boldsymbol{B}_2[1] = C_3$

$\exists B_2 : \boldsymbol{B}_2[2] = C_4$

$\exists B_2 : \boldsymbol{B}_2[3] = C_5$

and

$\exists B_3 : \boldsymbol{B}_3[1] = C_6$

$\exists B_3 : \boldsymbol{B}_3[2] = C_7$

and

$\exists B_4 : \boldsymbol{B}_4[1] = C_8$

$\exists B_4 : \boldsymbol{B}_4[2] = C_9$

$\exists B_4 : \boldsymbol{B}_4[3] = C_{10}$

and that the sequence has progressed to the stage where

$\boldsymbol{B}_1'[1] = \text{true}$

$\boldsymbol{B}_1'[2] = \text{true}$

$\boldsymbol{B}_2'[1] = \text{true}$

$B_2'[2]$ = true

$B_2'[3]$ = true

$B_3'[1]$ = true

$B_3'[2]$ = false

$B_4'[1]$ = x

$B_4'[2]$ = x

$B_4'[3]$ = x x = don't care state

The *security status* of the system can be determined at any time by applying the status function,

$$S\ (A_1,4) = S\ (B_1,2)\ \text{AND}\ S\ (B_2,3)\ \text{AND}\ S\ (B_3,2)\ \text{AND}\ S\ (B_4,3);$$

At the point of failure,

$S\ (B_1,2)$ = true,

$S\ (B_2,3)$ = true,

$S\ (B_3,1)$ = true, *BUT*

$S\ (B_3,2)$ = false

and, of course, $S\ (B_4,x)$ = false, but this is a 'don't care' state (indicated by 'x').

Therefore, by applying the recursive function S it is quickly determined that

$S\ (A_1,1)$ = true,

$S\ (A_1,2)$ = true, *BUT*

$S\ (A_1,3)$ = false

The position of the failure is therefore located at S $(A_1,3)$ which is the first *false* result. And since

$A_1[3] = B_3$ AND

$B_3[2] = C_7$

then C_7 is the failed atom.

Between each operation and the next, the CSS stores the result of the operation in the SMIB secure temporary data store. There is *fault protection* and *error recovery* built into the hardware/software to indicate failures. In practice, these will generally take the form of exception errors generated by the hardware, such as divide-by-zero interrupts or bounds violation interrupts. Error recovery allows the system to re-try operations, which might have failed due to a temporary problem such as power-supply transient. If on re-try the operation is successful then the sequence will continue. If, after a specified number of retrys, the error persists then the complete security service is aborted and the appropriate error flag return made to the calling application.

The notion of *timeliness* is very important in determining the correctness of the execution of the CSS function sequences. (Timeliness within the context of the formal analysis of protocols is discussed in section 6.9.2.). The protected real-time clock atom is used to time the execution rate of the sequences. In the event of inactivity or failure of a sequence which cannot be reinitiated by the recovery agent, the monitor agent will inform the security services agent of the failure. The external environment agent will then inform the calling process of CSS failure and initiate a shutdown of security services. These include a clearing of the CSS sequencing temporary data stores and the SMIB temporary data stores to ensure that confidential data such as keys and partially encrypted messages held in these stores cannot be externally accessed.

6.7. Examples of Sequences for Data Manipulation

Having established the general principle of the sequenced state machine, some examples of actual machines within the prototype CSS are now given.

Assume that we wish to use the CSS to implement a *non-repudiation of origin* service, an important security service recommended in ISO 7498-2. The *mechanisms* required to accomplish the security *service* of non-repudiation of origin might be designed as follows:

1. the sending party will need initially to generate an RSA key pair, placing the *public* key in the *directory* at the SMC, and keeping the *secret* key to himself in the local protected store in the SMIB;
2. the sender will need to hash the document he wishes to effect non-repudiation upon, using a public, system-wide hash function approved for the purpose. He will then sign the *digest* (the reduced data block resulting from a hash operation, similar to a CRC or checksum) with his secret key, including a time stamp to prevent replay attacks;
3. the sender will then send the document plus the signed digest (possibly via different routes) to the recipient;
4. the recipient will need to create a local copy of the document digest using the hash function;
5. the recipient then 'unsigns' the received (signed) digest by using the senders public key (which he has received from the directory and certified

by the SMC);

6. the recipient compares his local copy of the digest with the received digest he has just 'unsigned'.

If the digests match, then the document *must* have been sent by the purported sender, because nobody else would be able to create a digest using the senders secret key (unless the secret key had been compromised). The sender cannot therefore deny having sent the document. If the digests fail to match, the recipient can reject the document because either:

1. it was not sent by the legitimate sender, and hence the received, signed digest is incorrect;
2. it has been tampered with in transit, in which case the local digest will be incorrect.

6.7.1. Creating an RSA Key Pair

The *mechanism* of RSA key generation involves the following *components*:

1. the generation of three prime numbers, say p, q and e, by means of Rabin's algorithm;
2. determining the system modulus $m = pq$;
3. evaluation of the Totient function $\phi(pq)$ calculated from (p-1)(q-1);
4. determining the decryption quantity d by performing Euclid's algorithm such that $de \equiv 1 \pmod{\phi(pq)}$;
5. output the key pair $P_k = \{e,m\}$ and $S_k = \{d,m\}$.

These *components* in turn are broken down into sequences of *atoms*. The component designated 'Rabin's algorithm' is given as an example of the use of *atoms* in the construction of a component in Figure 6.15.

Figure 6.15. The Atoms used in Rabin's Algorithm for Primality Testing

```
FUNCTION Test_Prime (num : s255) : boolean;
begin
num_1 := n - 1;                        {SUBTRACT}
q := num_1;
k := 0;
WHILE q MOD 2 = 0 DO                   {DO WHILE LOOP}
      begin
      k := k + 1;                      {ADDITION}
      q := q DIV 2;                    {DIV}
      end;
Exponent_To_Binary (q);
FOR trial_count := 1 TO trials do
      { t passes of the algorithm = 1/4^t probability of failure }
      begin
      Finished := false;
      str (Primes [trial_count - 1],x);
      j := 0;
      y := Mod_Exp (x,n);              { MOD_EXP}
      REPEAT                           {REPEAT UNTIL LOOP}
```

```
            IF ( (j = 0) AND (y = 1) ) OR (y = num_1 ) THEN
                    begin
                    Test_Prime := true;  {GENERATE CONTROL BOOLEAN}
                    Finished := true;    {GENERATE CONTROL BOOLEAN}
                    end
            ELSE
            IF (j <> 0) AND (y = 1) THEN {OR}
                    begin
                    Test_Prime := false;          {GENERATE RETURN BOOLEAN}
                    Exit;                         {END}
                    end
            ELSE
            begin
            j := j + 1;                           {ADDITION}
            y := sqr (y) MOD n;                   {MULTIPLY and MOD}
            end;
        UNTIL (j = k) OR (Finished);
        IF (j = k) AND (NOT Finished) THEN
            begin
            Test_Prime := false;                  {GENERATE RETURN BOOLEAN}
            Exit;                                 {END}
            end;
        end;
end; □
```

The system modulus and its Totient function are calculated using the *multiply* atom. The decryption quantity is evaluated using the Euclid algorithm component, described in

Figure 6.16. Similarly, the file is hashed using the *repeated square MOD n* algorithm which in turn uses the MOD_EXP atom, and so forth.

Figure 6.16. The Atoms used in Euclid's Algorithm

```
FUNCTION GCD (n1,n2 : integer ) : integer;
begin
r := n1;
z := n2;
WHILE z < > 0 DO                    {DO WHILE LOOP}
        begin
        m := r MOD z;               {MOD}
        r := z;
        z := m;
        end;
GCD := r;                           {RETURN}
end;
```

6.7.2. Creating a QRC Key Pair

The public/secret key pair for the QRC is much simpler to create than the key pair required for RSA. Rabin's algorithm is used to generate a pair of prime numbers, both of which are *Blum integers* (that is, they are congruent to 3 mod 4). This prime number pair comprise the secret key, and their product, calculated using the multiply atom, comprises the public key.

6.7.3. RSA Block Encryption

RSA encryption just uses the MOD_EXP atom. The data is blocked according to the size of the modulus in use, and then each data block is raised to the power of the appropriate encrypting or decrypting exponent and reduced modulo the system modulus.

6.7.4. File Integrity Verification

File integrity verification uses the X.509 or similar hash function to create a *digest* of the file. This basically comprises a running block checksum of the data within the file. The file and its digest can then be sent as a pair to the recipient via independent routes. On receipt, the user also hashes the file to the digest and compares his result with the digest received with the file. If the two are identical, then he can assume that no modification of the file has taken place. If the digests differ, then he knows that the data within the file has been modified (either accidentally or deliberately) and he can reject the file. The method relies on the principle that it is computationally infeasible to construct two different files which would hash to the same digest.

In a similar way, the rest of the functions can be constructed from the mechanisms, components and atoms of the CSS.

6.8. Examples of Sequences for Protocol Generation

As well as generating mechanisms for data manipulation, the layered sequence methodology is also used to generate *protocols* for secure communication between agents. A protocol is defined as an algorithm for implementing a class of transactions. Protocols for even moderately sophisticated transactions involve a complex layering of processes and capabilities.

The principle technique for implementing such a capability is the cryptographic concealment of the message. Until quite recently, research efforts on secure protocols have concentrated on the security of the underlying cryptographic transformations. The need has arisen for new protocols, however, whose security properties are not readily apparent. In these protocols, it is not only the security of the cryptosystem that plays a role but also the logic of the implementation. Even given the *a priori* assumption that the cryptosystem offers perfect concealment, flaws in the protocol logic, when undetected, are as damaging to overall security as a compromised cryptosystem.

The protocol approach to communication offers a number of practical advantages. Firstly, protocols may be logically separated from the software and hardware which generate them. Secondly, protocols are generally high-level constructions which can be examined in detail. Thirdly, and of considerable importance in the CSS, protocols can be implemented using hardware or secure hardware-isolated software so that *all* communication channels are *explicit* and *observable*. This effectively rules out the possibility of *covert channels* discussed in section 4.2.2.

Two examples are given, an authentication protocol and a key distribution protocol. The formal proofs of these two protocols are given later in the chapter.

6.8.1. An Authentication Protocol

The authentication protocol used is due to Needham & Schroeder [NEED,1978].

The protocol definition comprises a sequence of five exchanges:

1. $\mathbf{A \rightarrow SMC}$: $\mathbf{A, B, N_a}$

2. $\mathbf{SMC \rightarrow A}$: $\mathbf{\{N_a, B, K_{ab}, \{K_{ab}, A\}_{Kbs}\}_{Kas}}$

3. $\mathbf{A \rightarrow B}$: $\mathbf{\{K_{ab}, A\}_{Kbs}}$

4. $\mathbf{B \rightarrow A}$: $\{N_b\}_{Kab}$

5. $\mathbf{A \rightarrow B}$: $\{f(N_b)\}_{Kab}$

Only **A** makes contact with the SMC including a timestamp $\mathbf{N_a}$, which then provides **A** with a conversation key $\mathbf{K_{ab}}$, together with a *certificate* $\{\mathbf{K_{ab}}, \mathbf{A}\}_{Kbs}\}$ encrypted with **B**'s key. This conveys the conversation key and **A**'s identity to **B**. **B** decrypts this certificate and carries out a handshake with **A** to be assured that **A** is *current*, since the certificate might be a replay! **A** then returns to **B** a function of the timestamp (which could take the form of a hash function, for example). In the following protocol, examples are given as to how the agents interact to accomplish the various steps. Complete descriptions of these interactions are lengthy, repetitive and tedious and so are not given explicitly at every step. Clearly, the sequence of interactions for decryption is almost identical to that for encryption apart from the use of a slightly different mechanism under the control of the SMA. Only those interactions of significance or which introduce a new agent are given to avoid repetition. Broken down, the protocol proceeds as follows:

1. A sends a cleartext message to the SMC containing a timestamp **N**:
   - (a) a request for a timestamp is passed across the API to the EEA;
   - (b) the EEA routes the control portion of the request to the SSA;
   - (c) the SSA instructs the SMA to generate a timestamp;
   - (d) the SMA controls the real-time clock to generate the time-stamp;
   - (e) the time-stamp is passed back to the calling process via the EEA;
   - (f) the OSI communication channel is used to communicate with the SMC.

   Note that there is no need for a cryptographic protocol involving the public key of the SMC since this message is *en clair*.

2. The reply from the SMC contains a conversation key $\mathbf{K}_{ab}$. The CSS generates a random conversation key using the random number generator component via a sequence of agent interactions similar to those described in (1) above. The reply is encrypted with the secret key $\mathbf{K}_{as}$. The encryption is performed by the following steps:
   - (a) a request for the message to be encrypted is passed across the API to the EEA;
   - (b) the EEA routes the control portion of the request to the SSA and the data portion (the message contents) to the CSS internal data bus;
   - (c) the SSA instructs the SMA to initiate an encryption;
   - (d) the SSA requests $\mathbf{K}_{as}$ from the SMIB via the SMIBA;
   - (e) the SMA controls the hardware to generate the encryption algorithm;
   - (f) the MA ensures that the sequencing for the encryption algorithm is carried out correctly;

(g) the encrypted data is passed back to the calling process via the EEA;

(f) the OSI communication channel is used to communicate with the **A**.

3. **A** then decrypts the reply using the decryption mechanism, and **A** knows the message is not a replay due to the timestamp.

4. **A** sends the part of the message intended for **B** to **B**.

5. **B** decrypts the message to also obtain the same conversation key as **A**, using the decryption mechanism.

6. **A** and **B** can now compare conversation keys to ensure that they are the same.

7. **A** and **B** now believe that

   (a) the other party exists currently;

   (b) the other party has sent their message recently;

   (c) both parties are in possession of the same key.

6.8.2. A Key Distribution Protocol

The key management algorithm used by the CSS to establish secure exchange of data between two peer entities is due to [RAMA,1990].

In the OSIRM architecture, REQUEST, INDICATION, RESPONSE and CONFIRM are the four basic service primitives used in the connection establishment, data transfer and connection release phases respectively, see section 3.3. Each primitive carries with it parameters to convey various pieces of information to its peer entity at the other end system. During the connection establishment phase, the two peer entities negotiate a set of parameters to be used during the data transfer stage. In the CSS key exchange algorithm, the security parameters are also negotiated between the end systems during this phase. In particular, the receipt of the public key $\mathbf{E_B}$ of system **B** and the generation and distribution of the session key (SK) are carried out during the connection establishment phase. The session key is then used to secure user data during the data transfer stage, and at the end of data transfer, the connection is disconnected and the session key destroyed.

As previously discussed, in a large system the SMC is responsible for the holding and distribution of valid public keys. The public key is only used for the exchange of a symmetric session key (such as DES) which is then used for the actual data security throughout the session due to greater efficiency than public key encryption. This is known as a *hybrid* encryption mechanism. To protect against the possibility that an impostor supplies the SMC with his own public key substituted for the genuine key of **A**, a *certificate* mechanism is used. In this mechanism, after registering the public key $\mathbf{E_A}$ of user **A** in the public key directory of the SMIB, the SMC sends the following certificate to user **A** in ciphertext form:

$$\text{SMC} \rightarrow \text{A}: \quad C_A = D_K\ (A,\ E_A,\ T)$$

where $\mathbf{C_A}$ is the certificate, $\mathbf{E_A}$ is a copy of the public key of **A** as received by the SMC, $\mathbf{D_K}$ is the secret key of the SMC, and **T** is a timestamp to ensure the freshness of the certificate against replay attack (see section 6.9.). User **A** decrypts the certificate using the public key of the SMC, and verifies that the copy of his public key as stored by the

SMC in the public key directory of the SMIB is in fact genuine. Examples of the mechanisms for the generation of the keys and the encryption/decryption are given in section 6.7.

The complete protocol as used by the CSS for the generation of public keys by users, the storage of the keys at the SMC, the distribution of public keys by the SMC and the exchange of symmetric session keys by users is now described. A formal proof of the security of this protocol can be found in section 6.9.6.

CERTIFICATION PHASE

1. User **A** generates his own public key by the *mechanism* described in section 6.7.1. He then transmits his public key in plain text to the SMC for registration and storage in the public key directory of the SMIB:

   $\mathbf{A} \rightarrow \mathbf{SMC}$: (A, E_A)

2. The **SMC** registers user **A**'s public key in the SMIB via the SMIBA and sends a *certificate* to **A** in enciphered form as described:

   $\mathbf{SMC} \rightarrow \mathbf{A}$: $C_A = D_K\ (A, E_A, T_{0A})$

3. User **A** deciphers the certificate and verifies that the SMC has registered the correct public key:

   $E_K\ \{D_K\ (A, E_A, T_{0A})\} = (A, E_A, T_{0A})$

4. Similarly, user **B** carries out steps 1 to 3 above:

B → **SMC**: (B, E_B)

SMC → **B**: $C_B = D_K$ (B, E_B, T_{0B})

E_K {D_K (B, E_B, T_{0B})} = (B, E_B, T_{0B})

CONNECTION PHASE

5. During the *connection establishment* phase, the service user at the $n+1$ layer at system **A** issues a CONNECT_REQUEST primitive to its service provider at the n layer, indicating that it wishes to establish a *secure* communication with remote system **B**.

6. The n layer at **A** requests the certificates $\mathbf{C_A}$ and $\mathbf{C_B}$ from the **SMC**:

A → **SMC**: (A,B)

and the **SMC** replies to **A** with:

SMC → **A**: {D_K (A, E_A, E_A(SK), T_1},

{D_K (B, E_B, E_B(SK), T_1}

= (C_A, C_B)

where $\mathbf{T_1}$ is now the timestamp indicating the current time at the **SMC** and SK is a randomly generated session key.

7. End system **A** decrypts the certificates with the public key of the **SMC** to obtain

the contents. He verifies his own public key as being correct, notes the public key of **B**, and further decrypts the session key with his own secret key to obtain SK in plaintext. Now in order to provide both *secrecy* and *authenticity*, user **A** enciphers the session key as well as the current timestamp T_2 using his own private key, encrypts the result with the public key of **B**, and sends the result to **B**:

$\mathbf{A} \rightarrow \mathbf{B}$: $X_{AB} = \{(SK, T_2)\ D_A\}\ E_B$

8. The *n* layer inserts the message (C_A, C_B, X_{AB}) into the variable part of the CONNECT-REQUEST (CR)-PDU using the parameter code, parameter length indication and parameter value fields as defined in the OSIRM, and transmits the resultant CR-PDU to the *n* layer of end system **B**.

9. The *n* layer at **B** decrypts $\mathbf{X_{AB}}$ using the information available in $\mathbf{C_A}$ and $\mathbf{C_B}$ and obtains the session key SK. System **B** then issues a CONNECT-INDICATION primitive to its *n*+1 layer.

10. After processing the indication primitive, the service user in the *n*+1 layer at system **B** now issues a CONNECT-RESPONSE primitive to its *n* layer.

11. The *n* layer at **B** inserts the security parameter (C_A, C_B, X_{AB}) into its variable part of the CONNECT-RESPONSE PDU and then transmits it to the *n* layer of system **A**:

$\mathbf{B} \rightarrow \mathbf{A}$: (C_A, C_B, X_{BA})

where X_{BA} is $E_A\{D_B(SK, T_3)\}$ and T_3 is the current time at **B**. It should be noted that X_{BA} is sent to **A** so that **A** can verify whether or not **B** has received the valid session key SK without modification. This also serves as an acknowledgement sent to **A** acknowledging the receipt of SK by **B**.

12. The *n* layer at **A** decrypts the received message from **B** and verifies whether or not **B** has received the valid session key SK. It then issues a CONNECT-CONFIRM primitive to its service user in the *n*+1 layer.

DATA TRANSFER PHASE

13. Since both systems **A** and **B** now share the session key SK they can encrypt the user data portion of a PDU. The service user at the *n*+1 layer issues a DATA(DT)-REQUEST primitive to its service provider at the *n* layer. Now the *n* layer at **A** encrypts the user data and transmits the resultant DATA(DT)-PDU to the *n* layer at **B**. On receipt, the *n* layer at **B** decrypts the user data portion of DT-PDU and issues a DATA(DT)-INDICATION primitive to its *n*+1 layer. In a similar fashion, **B** transmits its DT-PDU user data in encrypted form to **A**.

CONNECTION RELEASE PHASE

14. After data transfer, the *n*+1 layer at **A** (or **B**) issues a DISCONNECT-REQUEST (DR) primitive to its *n* layer. This DR primitive is transmitted to the *n* layer of the other system. The *n* layer then issues a DISCONNECT-INDICATION primitive to its *n*+1 layer. At the end of the session, SK is destroyed.

6.9. A Formal Proof of Protocol Sequence Security

In a distributed security environment, it is necessary to have procedures whereby various remote components of the system can communicate in a rigorous and secure manner. In the context of the CSS described, the agents of the CSS may be in physically remote locations, and yet need to convey information to each other concerning the security activities present on the network. These rigorous procedures are the communication *protocols*. Having established a methodology to prove the functional correctness of sequences built up from layers of finite state machines in a vertical sense, it remains to prove the security validity of the sequences themselves when used to generate the protocols which communicate between the agents in a distributed system.

Following Burrows, Abadi and Needham [BURR,1988], we use a formal logic to analyse the validity of the protocols between the communicating agents of the CSS.

6.9.1. A Formal Definition of Protocol Security

The definition of protocol security as used in this formal analysis depends on the principles of *belief* and *action*. If a subject is trusted in a security sense, then this is a statement of *belief* on behalf of the system that any *actions* carried out by that subject within the system will fall within the security policy of that system. If a subject is not trusted, however, then the system must ensure that the subject behaves according to the security policy, and this may be achieved by making untrusted subjects submit to the jurisdiction of trusted subjects. In the CSS, the trust is placed in the *Security Management Centres*, one of whose security functions is the supervision and enforcement of system security policy.

6.9.2. Requirements of Proofs about Protocols

Between agents of that part of the CSS which is completely contained within a local protected hardware unit, the need for many of the protocols is removed, because the data cannot be accessed or interfered with by an external influence. The design of the protected hardware suggested for use with the CSS does not allow access to any internal data bus, and in the event of physical violation, all internal data that would facilitate compromise of the system by the attacker is destroyed [WEIN,1987].

In a distributed computing system, however, the protocols between the distributed agents of the CSS must be highly resistant to compromise. A guarantee of *absolute security* is not possible, mainly due to the difficulty in adequately describing in formal terms what is meant by 'absolute security'. What *can* be said, is that given that certain subjects within the system are *trusted*, it is possible to develop protocols to ensure the extension of relative trust from these subjects to cover all the other subjects as required. The extension of trust in this way is called the development of a *trusted path* and extends the security perimeter to include as many subjects and objects as desired, see Chapter 2 and section 5.4.

The analysis of the protocols which provide the inter-agent communication in the CSS uses a formal logic specifically designed to define communication protocols in terms of security functionality. It is sufficiently rigorous to distinguish between the implementation of the protocol and the initial assumptions from which the protocol was developed, and to highlight properties of the protocol which may not be evident when initially conceived. These properties include such concerns as,

1. does the protocol achieve the goal?

2. if not, is it possible to modify or extend the protocol to achieve the goal?

3. what assumptions does the protocol require?

4. does the protocol assume more knowledge than is actually available when claiming to meet the goal?

5. does the protocol produce more knowledge than necessary to fulfil the goal?

6. does the protocol do anything unnecessary which could be omitted without weakening the goal, for example, does the protocol encrypt data which need not be encrypted?

7. does the protocol fail to do anything that is necessary to secure the goal, for example, does the protocol fail to encrypt critical data which needs encryption?

Concerns such as these are important because, as stated earlier, well intentioned but flawed policy can lead to anomalies which can result in security 'leaks'. These weaknesses can be very subtle indeed, and almost impossible to detect without a formal analysis. In particular, *covert channels* may have inadvertently resulted due to minor oversights in the design of the system. These channels may be exploited by a sophisticated attacker to leak information from the system, although the information bandwidth of such channels is usually very small. Refinement of the protocols using the logic can result in the elimination of many of these shortcomings.

The initial requirement of all the inter-agent protocols is the ability for the agents to convince each other about the identity of the peer entity in the association. In the simplest sense, this *authentication* is the guarantee that if the two entities are really who they claim to be then they will end up in possession of a shared secret which will allow then to communicate with *secrecy* and *integrity*. Achievement of these two goals will preclude the possibility of an attacker from either gaining information about the content of the communication or being able to influence its content.

A very important aspect which much of the literature does not address is the notion of *timeliness* of the protocol. If an identical protocol is used for every instance of a communication, then an attacker can simulate an genuine instance simply by *replaying* a previous communication, even if he has no idea whatever of its content. This is the classic *replay attack* scenario, which is very easy to mount in a computer communications environment. It is therefore very important to include the notion that a protocol must be *timely* in order to prevent attacks of this sort. In practical terms, this means the use of *time stamps* in all protocols that are vulnerable to replay attack, and the protected hardware real-time clock is included in the CSS for this purpose. Incorporating timeliness into the formalism, however, has proved difficult in previous attempts at logic of this sort. This analysis builds on the work of Burrows *et al* in the use of *nonces* which are expressions invented for the express purpose of being 'fresh'. As well as time stamps, other nonces can include the use of *random numbers*. If entity **A** invents a random number which has not previously been used in a communication and intimates this value to entity **B**, and entity **A** subsequently receives from **B** a message which fully includes the random number or a one-way function of it, then **A** can be assured that the message originated *after* the communication of the number to **B**, so long as the *integrity* of the message can be guaranteed.

Protocols are traditionally described by listing the messages sent between the communicating parties, showing in symbolic form the contents of the messages, the source, destination and any encrypting keys used. This conventional approach is not well suited to a formal analysis because the logic requires that an exact meaning be attached to certain elements of the messages, and this cannot always be inferred from the content. Each message is therefore translated into a logical formula before analysis, which is essentially an idealised form of the original message. *Assertions* are then made about each protocol in the same notation to describe the beliefs held by the various entities involved in the protocol.

6.9.3. The Basic Notation

Following Burrows *et al* [BURR,1988], the basic formalism is built on a many-sorted modal logic. We recognise three type of entity,

1. subjects;
2. encryption keys;
3. messages.

In general, the symbols **A**, **B**, and **S** denote specific subjects, $\mathbf{K}_{ab}$, $\mathbf{K}_{as}$, and $\mathbf{K}_{bs}$ denote specific keys and $\mathbf{N}_a$, $\mathbf{N}_b$, and $\mathbf{N}_c$ denote specific statements. The symbols **P** and **Q** range over subjects; **X** and **Y** range over statements; **K** ranges over encryption keys.

The only propositional connective is conjunction, denoted by a comma. Throughout, conjunctions are treated as sets.

In addition, the following are defined using a notation similar to that of Burrows *et al*:

P $\models$ **X**:	**P** *believes* **X** or is entitled to believe **X**. The subject **P** may then act as though **X** is true.
P $\sim$ **X**:	**P** *once said* **X**. The subject **P** sent a message at some unspecified time which included the statement **X**. It is not known whether the message is old (possibly a replay) or part of the current communication, but **P** believed **X** when he sent it.
P $\Rightarrow$ **X**:	P has *jurisdiction* over **X**. The subject **P** is an authority on **X** and should be trusted in all matters concerning **X**. For example, the SMC is trusted to supply authentic public keys.
P $\Leftrightarrow_K$ **Q**:	P and **Q** may use the valid key **K** to communicate.
$\{X\}_K$:	The statement **X** encrypted with the key **K**. We can extend this to $\{X\}_K$ *signed* **P**.
P $\triangleleft$ **X**:	**P** *sees* **X**. The subject **P** has received a message containing **X** and can read **X** (possibly after decryption). Clearly, **P** can repeat **X** in other messages.
#(X):	the statement **X** is *fresh*, that is, **X** has not been used in a message before the current protocol.

6.9.4. Formal Proofs of Authentication between Agents

As stated, the fundamental protocol which is always implemented between remote agents is an *authentication* protocol. (*Confidentiality* and *integrity* do not require formal protocols; they may be achieved by encryption alone.) In authentication, the fundamental concern is the distinction between *past* and *present*. The present *epoch* is deemed to start at the set-up time of the current protocol and all interactions later than this time are called *recent*. All communications before this time are deemed to be in the past, and the authentication protocol must be very careful to reject any of these past messages and not be subverted into believing them to be recent, and hence fall victim to a replay attack. In particular,

1. a belief about a past communication is *not* carried forward into the present one;

2. all beliefs held in the present are stable for the duration of the current protocol.

Now the logical postulates can be described:

The *message meaning* rule concerns beliefs about encrypted messages:

$$\mathbf{P} \models \mathbf{Q} \overset{}{\Leftrightarrow}_{\mathrm{K}} \mathbf{P},\ \ \mathbf{P} \triangleleft \{\mathbf{X}\}_{\mathrm{K}} \ \textit{implies that}\ \ \mathbf{P} \models \mathbf{Q} \sim \mathbf{X}$$

The rule says that if **P** believes that a key **K** is shared only with **Q**, and he sees the message **X** encrypted under **K**, then **P** is entitled to believe that **Q** once said **X**.

The *nonce verification* rule states that if a message is *recent* then the sender still believes in it:

$$\mathbf{P} \models \#(\mathbf{X}),\ \mathbf{P} \models \mathbf{Q} \sim \mathbf{X}\ \textit{implies that}\ \mathbf{P} \models \mathbf{Q} \models \mathbf{X}$$

This says that if **P** believes that **X** is recent and that **Q** once said **X** then **P** believes that **Q** said **X** recently, and hence **Q** believes **X**.

The *jurisdiction* rule states that if **P** believes that **Q** has jurisdiction over **X** then **P** trusts **Q** on the truth of **X**.

$$\mathbf{P} \models \mathbf{Q} \Rightarrow \mathbf{X},\ \mathbf{P} \models \mathbf{Q} \models \mathbf{X}\ \textit{implies that}\ \mathbf{P} \models \mathbf{X}$$

In addition, we can heuristically observe that **P** believes a set of statements if and only if **P** believes each individual statement separately. Hence, we have the following three additional rules:

$$\mathbf{P} \models \mathbf{X},\ \mathbf{P} \models \mathbf{Y}\ \textit{implies that}\ \mathbf{P} \models (\mathbf{X},\mathbf{Y})$$

$$\mathbf{P} \models (\mathbf{X},\mathbf{Y})\ \textit{implies that}\ \mathbf{P} \models \mathbf{X}$$

$$\mathbf{P} \models \mathbf{Q} \models (\mathbf{X},\mathbf{Y})\ \textit{implies that}\ \mathbf{P} \models \mathbf{Q} \models \mathbf{X}$$

And similarly with the operator $\sim$:

$$\mathbf{P} \models \mathbf{Q} \sim (\mathbf{X},\mathbf{Y})\ \textit{implies that}\ \mathbf{P} \models \mathbf{Q} \sim \mathbf{X}$$

If a subject sees a message, then he also knows the content statements, providing that if the message is encrypted, he knows the keys,

$$\mathbf{P} \triangleleft (\mathbf{X},\mathbf{Y}) \textit{ implies that } \mathbf{P} \triangleleft \mathbf{X}$$

$$\mathbf{P} \models \mathbf{Q} \Leftrightarrow_K \mathbf{P},\ \mathbf{P} \triangleleft \{\mathbf{X}\}_K \textit{ implies that } \mathbf{P} \triangleleft \mathbf{X}$$

By virtue of the *message digest* which ensures that no part of a message can be altered without discovery, if one statement of a message is known ~~the~~ to be fresh, then the whole message must be fresh,

$$\mathbf{P} \models \#(\mathbf{X}) \textit{ implies that } \mathbf{P} \models \#(\mathbf{X},\mathbf{Y})$$

6.9.5. Idealisation of the CSS Protocols

Conventionally, each step of a protocol is written in the form

$$\mathbf{P} \rightarrow \mathbf{Q} : \textit{message}$$

which denotes that the subject **P** sends a message and that subject **Q** receives it. (Within the context of the CSS, confirmation of receipt may be assumed due to supervision by the CSS of possible disruption due to *denial of service* attacks against the security traffic on the network, discussed in detail elsewhere.) In order to express the *meaning* of the message, it is necessary to transform it into an idealised form which expresses the *semantics* of the message. For example **A** may wish to communicate a symmetric session key to **B** for future use, using a public key encryption transfer under the assumption that only **B** knows the secret key $\mathbf{K_{bs}}$,

$$\mathbf{A} \rightarrow \mathbf{B} : \{\mathbf{K_{ab}} \text{ is to be used for communication between us}\}$$

In terms of ideal formulation, this would be written

$$\mathbf{A} \rightarrow \mathbf{B} : \{\mathbf{A} \Leftrightarrow_{\mathbf{Kab}} \mathbf{B}\}_{\mathbf{Kbs}}$$

When the message arrives at **B**, we can infer that **B** *sees* the message

$$\mathbf{B} \triangleleft \{\mathbf{A} \Leftrightarrow_{\mathbf{Kab}} \mathbf{B}\}_{\mathbf{Kbs}}$$

and hence **B** becomes aware of the content of the message and can act upon it.

The analysis of protocols is carried out in four steps:

1. the idealised protocol is derived from the original idea;

2. assumptions are made about the initial state;

3. logical expressions are attached to protocol statements in the form of assertions about the state of the protocol after each statement;

4. the logical postulates are repeatedly applied to the assertions in order to discover the beliefs held by the parties to the protocol on completion.

The whole procedure may be repeated as many times as required, as new assumptions are found to be necessary to ensure that the correct beliefs end up being held by the parties involved.

Burrows *et al* have suggested the following three rules for the derivation of legal annotations to protocols,

1. for *single* protocol steps, the annotation

$$[Y](P \rightarrow Q : X)[Y, Q \triangleleft X]$$

   is legal. All beliefs held before a message is sent hold afterwards, the only difference is that the recipient *sees* the message;

2. for *sequences* of protocol steps, if

$$[X]S_1 \ldots . [Y] \text{ and } [Y]S_1' \ldots . [Z]$$

   are legal, then so is

$$[X]S_1 \ldots . [Y]S_1' \ldots . [Z]$$

   Thus, annotations can be concatenated;

3. the logical postulates used are

   (a) if X is an assertion (but not an assumption) in a legal annotation

$\mathcal{P}$, **X**′ is provable from **X**, and $\mathcal{P}'$ is the result of substituting **X**′ for **X** in $\mathcal{P}$, then $\mathcal{P}'$ is a legal annotation. Thus, new assertions can be derived from existing ones;

(b) if **X** is the assumption of a legal annotation $\mathcal{P}$, **X**′ is provable from **X**, and $\mathcal{P}'$ is the result of substituting (**X**,**X**′) for **X** in $\mathcal{P}$, then $\mathcal{P}'$ is a legal annotation. Thus, the *consequences* of the original assumptions can be written explicitly next to the original assumptions.

Burrows *et al* are careful to point out that the power of this method stems from the ability to follow the evolution from the original assumptions to the *consequences* of those assumption in a rigorous manner, in other words, we can formally demonstrate the validity of final beliefs about the state of a system from the initial beliefs.

As stated, the fundamental component of all the inter-agent protocols is authentication. Formally, authentication is complete between agent **A** and agent **B** if there is a **K** such that

$$\mathbf{A} \models \mathbf{A} \overset{}{\Leftrightarrow}_{\mathbf{K}} \mathbf{B}$$

$$\mathbf{B} \models \mathbf{A} \Leftrightarrow_{\mathbf{K}} \mathbf{B}$$

$$\mathbf{A} \models \mathbf{B} \models \mathbf{A} \Leftrightarrow_{\mathbf{K}} \mathbf{B}$$

$$\mathbf{B} \models \mathbf{A} \models \mathbf{A} \Leftrightarrow_{\mathbf{K}} \mathbf{B}$$

While the first two of the above are essential, the last two results are no stronger than merely desirable.

In their paper, Burrows *et al* go on to give two examples of the application of their logic, and their results are worth repeating here. Initially, they consider the Otway & Rees authentication protocol [OTWA,1987]. Adopting the following symbology:

A,B	principal subjects
K_{as},K_{bs}	symmetric keys **A** ⇔ server, **B** ⇔ server
N_a,N_b,M	nonces
K_{ab}	secret session key **A** ⇔ **B**
S	authentication server (in our context, the SMC)

Otway & Rees defined their protocol as comprising four messages:

1. **A → B:** $M, A, B, \{N_a, M, A, B\}_{Kas}$

2. **B → S:** $M, A, B, \{N_a, M, A, B\}_{Kas}, \{N_b, M, A, B\}_{Kbs}$

3. **S → B:** $M, \{N_a, K_{ab}\}_{Kas}, \{N_b, K_{ab})_{Kbs}$

4. **B → A:** $M, \{N_a, K_{ab}\}_{Kas}$

The protocol requires that **A** passes to **B** some encrypted information only useful to the server (SMC), together with enough information for **B** to make up a similar message. **B** forwards both to the server, who decrypts the messages and checks that **M**, **A**, **B** match in the encrypted parts. If so, the SMC generates the session key $\mathbf{K_{ab}}$ and embeds it into two messages, one for **A** and one for **B**, accompanied by the appropriate nonces. Both are sent to **B** who forwards **A**'s part on to **A**. Then **A**, **B** decrypt their respective messages, and if satisfied, proceed to use the key $\mathbf{K_{ab}}$.

Transforming to the logic and using the abbreviation $\mathbf{N_c}$ in place of **M,A,B**, we rewrite the protocol as follows

1. $\mathbf{A \rightarrow B}$: $\{N_a, N_c\}_{K_{as}}$

2. $\mathbf{B \rightarrow S}$: $\{N_a, N_c\}_{K_{as}}, \{N_b, N_c\}_{K_{bs}}$

3. $\mathbf{S \rightarrow B}$: $\{N_a, (A \overset{K_{ab}}{\Leftrightarrow} B), (B \sim N_c)\}_{K_{as}}, \{N_b, (A \overset{K_{ab}}{\Leftrightarrow} B), (A \sim N_c)\}_{K_{bs}}$

4. $\mathbf{B \rightarrow A}$: $\{N_a, (A \overset{K_{ab}}{\Leftrightarrow} B), (B \sim N_c)\}_{K_{as}}$.

The assumptions made by the protocol are

$A \models A \overset{K_{as}}{\Leftrightarrow} S$

$B \models B \overset{K_{bs}}{\Leftrightarrow} S$

$S \models A \overset{K_{as}}{\Leftrightarrow} S$

$S \models B \overset{K_{bs}}{\Leftrightarrow} S$

$S \models A \overset{K_{ab}}{\Leftrightarrow} B$

$A \models (S \Rightarrow A \overset{K_{ab}}{\Leftrightarrow} B)$

$B \models (S \Rightarrow A \overset{K_{ab}}{\leftrightarrow} B)$

$A \models (S \Rightarrow B \sim X)$

$B \models (S \Rightarrow A \sim X)$

$A \models \#(N_a)$

$B \models \#(N_b)$

$A \models \#(N_c)$

The logical analysis now proceeds as follows:

1. $A \rightarrow B$: **B** sees the message from **A** but cannot read it

$$B \triangleleft \{N_a, N_c\}_{K_{as}}$$

2. **B** generates a message of the same form, and passes both to **S**.

3. S decrypts each part by the *message meaning postulate*, and can deduce that **A** and **B** have encrypted the nonce N_c in their packets.

$$S \models A \sim (N_a, N_c), \qquad S \models B \sim (N_b, N_c)$$

NOTE that S cannot tell if this is a replay or not since nothing in the message tells him if the message is fresh!

4. S emits a message containing two encrypted statements to **B**.

5. One part of the message is intended for **A**, and **B** passes this on. At this point, **A** and **B** have received a message from the SMC with a new session key plus a nonce. **A** and **B** successively apply the

(a) message meaning

(b) nonce verification

(c) jurisdiction

postulates to emerge with the final beliefs,

$\mathbf{A} \models (\mathbf{A} \Leftrightarrow_{K_{ab}} \mathbf{B})$

$\mathbf{A} \models (\mathbf{B} \models N_c)$

$\mathbf{B} \models (\mathbf{A} \Leftrightarrow_{K_{ab}} \mathbf{B})$

$\mathbf{B} \models (\mathbf{A} \models N_c)$

NOTE that the logic has exposed this as being a far from complete or reliable authentication protocol. It could be completed by handshaking between **A** and **B**, but the weakness is that the session key $\mathbf{K}_{ab}$ is never used by either party during the protocol, and so neither party can be assured that the other is in possession of the same key. In fact, **A** is in a slightly stronger position than **B**, in that **A** has been told that **B** emitted a message containing a nonce that **A** believes to be fresh. This allows **A** to infer that **B** sent the message recently, that is, **B** currently exists in the system. **B**, however, has only been told by the SMC that **A** has used a nonce, but has no idea if this is a replay attack or not!

It is interesting to note that, in addition to the weaknesses exposed, the analysis reveals that the protocol also contains some redundancy:

1. two nonces are put up by **A**, but one would suffice. The verification using $\mathbf{N_a}$ is just as easily done by $\mathbf{N_c}$ and hence the nonce $\mathbf{N_a}$ is redundant;

2. $\mathbf{N_b}$ in the second message need not be encrypted.

The above analysis leads to a quite different approach to the authentication problem. The following analysis concerns the Needham & Schroeder protocol [NEED,1978].

The concrete protocol definitions comprise a sequence of five messages:

1. $A \rightarrow S$: A, B, N_a

2. $S \rightarrow A$: $\{N_a, B, K_{ab}, \{K_{ab}, A\}_{Kbs}\}_{Kas}$

3. $A \rightarrow B$: $\{K_{ab}, A\}_{Kbs}$

4. $B \rightarrow A$: $\{N_b\}_{Kab}$

5. $A \rightarrow B$: $\{f(N_b)\}_{Kab}$

Only **A** makes contact with the SMC, who provides **A** with a session key $\mathbf{K_{ab}}$ together with a *certificate* encrypted with **B**'s key conveying the session key and **A**'s identity to **B**. **B** decrypts this certificate and carries out a nonce handshake with **A** to be assured that **A** is *current*, since the certificate might be a replay! **A** then returns to **B** a function of the nonce (which could take the form of a hash function, for example).

Transforming to the logic and using the same notation and substitutions as previously,

1. $\mathbf{A} \rightarrow \overset{S}{\cancel{\mathbf{B}}}$: $\mathbf{N_a}$

2. $\mathbf{S} \rightarrow \mathbf{A}$: $\{\mathbf{N_a}, (\mathbf{A} \Leftrightarrow_{K_{ab}} \mathbf{B}), \#(\mathbf{A} \Leftrightarrow_{K_{ab}} \mathbf{B}), \{\mathbf{A} \Leftrightarrow_{K_{ab}} \mathbf{B}\}_{K_{bs}}\}_{K_{as}}$

3. $\mathbf{A} \rightarrow \mathbf{B}$: $\{\mathbf{A} \Leftrightarrow_{K_{ab}} \mathbf{B}\}_{K_{bs}}$

4. $\mathbf{B} \rightarrow \mathbf{A}$: $\{\mathbf{N_b}, (\mathbf{A} \Leftrightarrow_{K_{ab}} \mathbf{B})\}_{K_{ab}}$ *signed* **B**

5. $\mathbf{A} \rightarrow \mathbf{B}$: $\{\mathbf{N_b}, (\mathbf{A} \Leftrightarrow_{K_{ab}} \mathbf{B})\}_{K_{ab}}$ *signed* **A**

The additional statements about the key $\mathbf{K_{ab}}$ in (2), (4) and (5) are there to assure **A** that the key can be used as a nonce, and to assure **A** and **B** that the other believes that the key is good. (These statements can be included because neither message would have been sent in the first place had these beliefs not been held).

The assumptions are

$\mathbf{A} \models \mathbf{A} \Leftrightarrow_{K_{as}} \mathbf{S}$

$\mathbf{B} \models \mathbf{B} \Leftrightarrow_{K_{bs}} \mathbf{S}$

$\mathbf{S} \models \mathbf{A} \Leftrightarrow_{K_{as}} \mathbf{S}$

$\mathbf{S} \models \mathbf{B} \Leftrightarrow_{K_{bs}} \mathbf{S}$

$S \models A \Leftrightarrow_{Kab} B$

$A \models (S \Rightarrow A \Leftrightarrow_{Kab} B)$

$B \models (S \Rightarrow A \Leftrightarrow_{Kab} B)$

$A \models (S \Rightarrow \#(A \Leftrightarrow_{Kab} B))$

$A \models \#(N_a)$

$B \models \#(N_b)$

$S \models \#(A \Leftrightarrow_{Kab} B)$

The logical analysis now proceeds as follows:

1. A sends a cleartext message containing a nonce. This is seen by the SMC who repeats the nonce in reply.

2. The reply from the SMC also contains the session key K_{ab}.

3. A then sees the entire message

$$A \triangleleft \{N_a, (A \Leftrightarrow_{Kab} B), \#(A \Leftrightarrow_{Kab} B), \{A \Leftrightarrow_{Kab} B\}_{Kbs}\}_{Kas}$$

   which A decrypts using the message meaning postulate.

4. Since A knows N_a to be fresh, he can apply the nonce verification postulate, giving

$$A \models S \models A \Leftrightarrow_{Kab} B, \quad A \models S \models \#(A \Leftrightarrow_{Kab} B)$$

5. The jurisdiction postulate allows **A** to infer

$$\mathbf{A} \models \mathbf{A} \overset{}{\Leftrightarrow}_{K_{ab}} \mathbf{B}, \quad \mathbf{A} \models \#(\mathbf{A} \Leftrightarrow_{K_{ab}} \mathbf{B})$$

6. Also, **A** has seen part of the message encrypted under **B**'s key

$$\mathbf{A} \triangleleft \{\mathbf{A} \Leftrightarrow_{K_{ab}} \mathbf{B}\}_{K_{bs}}$$

This allows **A** to safely send the message to **B**.

7. **B** can use the message meaning postulate and decrypt

$$\mathbf{B} \models \mathbf{S} \sim \mathbf{A} \Leftrightarrow_{K_{ab}} \mathbf{B}$$

Unlike **A**, however, **B** cannot proceed without resort to the dubious assumption that

$$\mathbf{B} \models \#(\mathbf{A} \Leftrightarrow_{K_{ab}} \mathbf{B})$$

The assumption is dubious because **B** knows of nothing in the message which is fresh, and so cannot tell when it was generated. **B** has to assume that the message from the SMC is fresh.

8. Using the extra assumption in (7), **B** can obtain the key

$$\mathbf{B} \models \mathbf{A} \Leftrightarrow_{K_{ab}} \mathbf{B}$$

via the nonce verification and jurisdiction postulates.

9. The last two messages cause **A** and **B** to believe that

(a) the other party exists currently,

(b) the other party has sent their message recently,

(c) both parties are in possession of the same key.

B first encrypts his nonce and sends it to **A**, who can infer that **B** believes in the key

$$\mathbf{A} \models \mathbf{B} \models \mathbf{A} \overset{}{\Leftrightarrow}_{K_{ab}} \mathbf{B}$$

because he has been guaranteed of the freshness of the key by the SMC.

10. **A** then replies similarly, and **B** can infer that **A** believes in the key

$$\mathbf{B} \models \mathbf{A} \models \mathbf{A} \Leftrightarrow_{K_{ab}} \mathbf{B}$$

NOTE that the freshness of the nonce $\mathbf{N}_b$ is sufficient for **B** to infer this. It is not necessary to re-use the dubious assumption made in (7).

11. The final beliefs with which both parties emerge are

$\mathbf{A} \models (\mathbf{A} \Leftrightarrow_{K_{ab}} \mathbf{B})$

$\mathbf{B} \models (\mathbf{A} \Leftrightarrow_{K_{ab}} \mathbf{B})$

$\mathbf{A} \models \mathbf{B} \models (\mathbf{A} \Leftrightarrow_{K_{ab}} \mathbf{B})$

$\mathbf{B} \models \mathbf{A} \models (\mathbf{A} \Leftrightarrow_{K_{ab}} \mathbf{B})$

In fact, we obtain the even stronger

$$B \models A \models B \models (A \overset{K_{ab}}{\Leftrightarrow} B)$$

This is a strong result, but is obtained at the expense of the extra assumption in (7).

6.9.6. Formal Proof of the CSS Key Distribution Protocol

A formal proof using the logic is now given for the key distribution protocol described in section 6.8.2. The symbology adopted is now:

A,B	principal subjects
S	security management centre
$\mathbf{E_A,E_B}$	the public keys of **A** and **B** respectively
$\mathbf{C_A,C_B}$	certificates of the public keys of **A** and **B** respectively
$\mathbf{D_A,D_B}$	the secret keys of **A** and **B** respectively
$\mathbf{E_K,D_K}$	the public and secret keys of the SMC respectively
SK	the session key
$\mathbf{T_{XX}}$	nonces

The unadorned protocol takes the form:

1. $A \rightarrow S$: (A, E_A)

2. $S \rightarrow A$: $C_A = D_K\ (A,\ E_A,\ T_{0A})$

3. $B \rightarrow S$: $(B,\ E_B)$

4. $S \rightarrow B$: $C_B = D_K\ (B,\ E_B,\ T_{0B})$

5. $A \rightarrow S$: (A,B)

6. $S \rightarrow A$: $\{D_K\ (A,\ E_A,\ E_A(SK),\ T_1\},$

$\{D_K\ (B,\ E_B,\ E_B(SK),\ T_1\}$

$= (C_A,\ C_B)$

7. $A \rightarrow B$: $X_{AB} = \{(SK,\ T_2)\ D_A\}\ E_B$

8. $B \rightarrow A$: $(C_A,\ C_B,\ X_{BA})$

Transforming the logic as before, the protocol becomes:

1. $A \rightarrow S$: $\{A,\ A \Leftrightarrow_{EA} S\}$

2. $S \rightarrow A$: $C_A = \{A,\ \#(A \Leftrightarrow_{EA} S)\}_{DK}$

3. $B \rightarrow S$: $\{B,\ B \Leftrightarrow_{EB} S\}$

4. $S \rightarrow B$: $C_B = \{B,\ \#(B \Leftrightarrow_{EB} S)\}_{DK}$

5. $A \rightarrow S$: (A,B)

6. $S \rightarrow A$: $\{\{A, \#(A \Leftrightarrow_{EA} S), \#(A \Leftrightarrow_{SK} B)\},$

$\{B, \#(B \Leftrightarrow_{EB} S), \#(A \Leftrightarrow_{SK} B)\}\}_{DK}$

7. $A \rightarrow B$: $\{\#(A \Leftrightarrow_{SK} B)_{DA}\}_{EB}$

8. $B \rightarrow A$: $\{\#(A \Leftrightarrow_{SK} B)_{DB}\}_{EA}$

The assumptions at the start of the protocol are:

$A \models A \Leftrightarrow_{EA,DA,EK,DK} S$

$B \models B \Leftrightarrow_{EB,DB,EK,DK} S$

$S \models A \Leftrightarrow_{EA,DA,EK,DK} S$

$S \models B \Leftrightarrow_{EB,DB,EK,DK} S$

$S \models A \Leftrightarrow_{SK} B$

$A \models (S \Rightarrow A \Leftrightarrow_{SK} B)$

$B \models (S \Rightarrow A \Leftrightarrow_{SK} B)$

$A \models (S \Rightarrow \#(A \Leftrightarrow_{SK} B))$

$B \models (S \Rightarrow \#(A \Leftrightarrow_{SK} B))$

$S \models \#(T_{0A})$

$S \models \#(T_{0B})$

$S \models \#(A \Leftrightarrow_{SK} B)$

The analysis of the protocol proceeds as follows:

CERTIFICATION PHASE

1. **A** transmits his own identity and public key in plain text to the **SMC**

   $\mathbf{A \rightarrow S}$: $\{\mathbf{A}, \mathbf{A} \Leftrightarrow_{EA} \mathbf{S}\}$

2. This is seen by the **SMC** which sends a certificate to **A** in reply:

   $\mathbf{S \rightarrow A}$: $\mathbf{C_A} = \{\mathbf{A}, \#(\mathbf{A} \Leftrightarrow_{EA} \mathbf{S})\}_{DK}$

3. User **A** sees the message

   $\mathbf{A} \triangleleft \{\mathbf{A}, \#(\mathbf{A} \Leftrightarrow_{EA} \mathbf{S})\}_{DK}$

   which **A** deciphers using the message meaning postulate. He also applies the nonce verification postulate which allows **A** to infer that the **SMC** has registered the correct public key, and the registration is fresh.

4. Similarly, user **B** carries out steps 1 to 3 above:

   $\mathbf{B \rightarrow S}$: $\{\mathbf{B}, \mathbf{B} \Leftrightarrow_{EB} \mathbf{S}\}$

   $\mathbf{S \rightarrow B}$: $\mathbf{C_B} = \{\mathbf{B}, \#(\mathbf{B} \Leftrightarrow_{EB} \mathbf{S})\}_{DK}$

   $\mathbf{B} \triangleleft \{\mathbf{B}, \#(\mathbf{B} \Leftrightarrow_{EB} \mathbf{S})\}_{DK}$

CONNECTION PHASE

5. A requests the certificates C_A and C_B from the SMC:

$A \rightarrow S$: (A,B)

6. The SMC replies to A with:

$S \rightarrow A$: $\{\{A, \#(A \Leftrightarrow_{EA} S), \#(A \Leftrightarrow_{SK} B)\},$
$\{B, \#(B \Leftrightarrow_{EB} S), \#(A \Leftrightarrow_{SK} B)\}\}_{DK}$

7. A sees the entire message:

$A \triangleleft \{\{A, \#(A \Leftrightarrow_{EA} S), \#(A \Leftrightarrow_{SK} B)\},$
$\{B, \#(B \Leftrightarrow_{EB} S), \#(A \Leftrightarrow_{SK} B)\}\}_{DK}$

and applies the message meaning postulate to decipher the contents, the nonce verification postulate to infer that the certificates came recently from the SMC, and the jurisdiction postulate to infer the freshness of the session key. Hence

$A \models S \sim \#(C_A, C_B)$
$A \models \#(A \Leftrightarrow_{SK} B)$

8. Now in order to provide both *secrecy* and *authenticity*, A enciphers the session key as well as the current nonce T_2 using his own private key, and encrypts the result with the public key of B, and sends the result to B:

$\mathbf{A} \rightarrow \mathbf{B}$: $\{\#(\mathbf{A} \Leftrightarrow_{SK} \mathbf{B})_{DA}\}_{EB}$

9. **B** then sees the message

$\mathbf{B} \triangleleft \{\#(\mathbf{A} \Leftrightarrow_{SK} \mathbf{B})_{DA}\}_{EB}$

which he decrypts using the message meaning postulate. He now sees the contents

$\mathbf{B} \triangleleft \{\#(\mathbf{A} \Leftrightarrow_{SK} \mathbf{B})_{DA}\}$

and applies a further decryption. Application of the message meaning postulate and nonce verification postulate allow **B** to infer that the message originated from **A** and that it is recent.

10. ~~A~~ B echoes the contents back to ~~B~~ A

$\mathbf{B} \rightarrow \mathbf{A}$: $\{\#(\mathbf{A} \Leftrightarrow_{SK} \mathbf{B})_{DB}\}_{EA}$

11. **A** then sees the message

$\mathbf{A} \triangleleft \{\#(\mathbf{A} \Leftrightarrow_{SK} \mathbf{B})_{DB}\}_{EA}$

which he decrypts using the message meaning postulate. He now sees the contents

$\mathbf{A} \triangleleft \{\#(\mathbf{A} \Leftrightarrow_{SK} \mathbf{B})_{DB}\}$

and applies a further decryption. Application of the message meaning postulate and nonce verification postulate allow **A** to infer that the message originated from **B** and that it is recent.

DATA TRANSFER PHASE

12. The final beliefs with which **A** and **B** emerge are

$\mathbf{A} \models (\mathbf{A} \Leftrightarrow_{SK} \mathbf{B})$

$\mathbf{B} \models (\mathbf{A} \Leftrightarrow_{SK} \mathbf{B})$

$\mathbf{A} \models \mathbf{B} \models (\mathbf{A} \Leftrightarrow_{SK} \mathbf{B})$

$\mathbf{B} \models \mathbf{A} \models (\mathbf{A} \Leftrightarrow_{SK} \mathbf{B})$

$\mathbf{A} \models \mathbf{S} \sim \#(\mathbf{A} \Leftrightarrow_{SK} \mathbf{B})$

Once the public keys are registered, the protocol will normally be initiated by one party, say **A**, at the connection phase. Note that in a similar manner to the two authentication protocols previously analysed, **B** is again not in as strong a position as **A** in that **B** cannot be as certain of either the freshness of the session key, or that the SMC generated it. **B** is totally reliant on **A** for his information, and at no stage makes contact with the SMC himself. This weakness is not exposed in the analysis in [RAMA,1990].

13. Since **A** and **B** now share the session key SK they can encrypt the user data portions of their PDUs.

The analysis of the other protocols used by the CSS is carried out in an identical manner.

7. CONCLUSIONS

7.1. A Review of Achievements of the Research

This thesis presents new conceptual and practical work in the following areas:

1. The aims and objectives set out in Chapter 1 have been achieved, and a demonstration distributed security system implemented using the principles developed during this research. A *Comprehensive Security System (CSS)* has been developed which conforms to many of the recommendations of the ISO Basic Reference Model - Security Architecture [ISO,7498-2], including:

   (a) implementation of the five basic security services of confidentiality, integrity, access control, authentication and detection of denial of service;

   (b) conceptualisation of the CSS as an Application Layer entity. While the prototype implements the majority of the security facilities at the Application Layer in accordance with recommendations, implementation at any of the seven layers is possible due to the vertical nature of the CSS Application Program Interface (API);

(c) the capability of the system to be fitted retrospectively to existing systems as a value-added service, providing that applications are written/rewritten to take advantage of the facilities offered by the CSS;

(d) the logical division of the system into *agents*, which allows the system to be implemented in a distributed architecture environment.

2. The system is capable of providing security services in a flexible and efficient manner due to the provision of all the required security services by a common set of *security mechanisms*. These are realised as a hierarchical structure of finite state machines, sequenced according to data stored in the *Security Management Information Base (SMIB)* of the CSS. At the lowest level, data is manipulated by hardware/software performing only five basic mathematical operators (+,-,*,mod,div). In this way, a wide range of services can be provided, with little or no constraint on future expansion of services or modification of encryption algorithms.

3. A new security analysis methodology has been devised based upon:

(a) a *monitored sequencing* of the layers of finite state machines which generate the security mechanisms and inter-agent communication protocols required to fully implement the security functionality in a rigorous manner. The approach uses a recursive algorithm for the determination and verification of the state of the CSS sequences at any given moment together with the notion of time-constraint to determine whether the system is subject to failure or possible attack by submission of invalid data;

(b) a *modified Hoare logic* for the verification of the protocols between agents of the CSS. The method develops the concepts of analysis through application of logical postulates to statements concerning *beliefs* about the global state of the system as a result of *actions* performed by subjects on objects within the system. In particular, the notion of timeliness is incorporated into the logic to cope with the possibility of replay attacks which are relatively easy to mount in an electronic communications environment.

4. The analysis methodology is particularly relevant to the very latest European initiatives under the COSINE Eureka project, which involves the development and proving of security mechanisms within the OSI distributed processing environment.

5. The concept of 'Security Management Centres' is introduced, which function as *supervisory* nodes for all security related traffic on the network. Their functions include supply and certification of public keys, supervision of peer entity associations, and session supervision to detect *denial of service* attacks against the system. In the event of such an attack being detected, the supervisory action may be to terminate the session or to determine the possibility of alternate connections.

6. A practical implementation of many of the new ideas has been produced, running on an Ethernet LAN of IBM Personal Computers.

7.2. Limitations of the Research

Although this research has gone some way to bringing about the realisation of the Extended OSI Security Architecture discussed in [MUFT,1992], several limitations remain:

1. The problem of *denial of service* has not been overcome. The CSS is capable of *detecting* possible instances of such an attack and taking such action as necessary to clear down secure associations in a controlled manner but it cannot *prevent* such an attack from taking place. It is difficult to see how this could ever be achieved in practice.

2. The formal logic used for the analysis of the protocols between agents of the CSS relies, like all formal methods, on the fundamental truth of the axioms. In the case of the CSS, the axioms are the initial beliefs about the state of the system. If these are incorrect then the conclusions drawn as a result of the logic will be in error. The initial verification of the functional correctness of a system *before* modification by protocol activity is difficult. The existing software verification methods described in Chapter 4 are concerned mainly with *modification* of data by procedures. Verification of the *initial* state is assumed in most arguments without justification.

7.3. Suggestions for Future Work

Although a practical implementation of the CSS has been produced, many of the ideas presented in this thesis are more of a proposal than a description of proven technology. While the basic tools of encryption are well understood, the application of these principles to large systems is very much in its infancy. Considerable effort will be required in defining an appropriate security model for a distributed system comprising multiple security domains and in defining the different servers that will be required to support such a distributed architecture. It is hoped that the work described here will show what can be achieved, but it represents only one possible approach to the problem.

The demonstration system comprises of one SMC and two hosted users. The work on the prototype CSS could be extended to a full software implementation for trials on a large scale network such as JANET. In addition, a complete hardware implementation of the local CSS processor including cryptographic hardware would be useful in demonstrating the strength of a full implementation. After the completion of the research described in this thesis, the use of the *Cryptech* PC Crypto Processor, a general-purpose number theory engine, is being investigated as a means of speeding up the cryptographic algorithms to a more realistic data throughput. The manufacturers claim a processing speed in excess of 12,600 bits per second for RSA encryption using a 512 bit modulus. Since the functions (addition, multiplication and modulo division) are individually accessible, however, it should be possible to program the card to implement the QRC cipher discussed in section 5.6. with possibly even higher data rates than for RSA. It is interesting to note that the American National Institute of Standards (NIST) have recently decided against adopting RSA, and this could give a boost to the importance of other public key ciphers such as QRC described in this research.

One of the major problems in rigorously predicting, enforcing and analysing the behaviour of software is that systems are increasing in complexity faster than new techniques are developed to cope with them. The fields of *safety-critical* and *fault-tolerant* computing, however, may have something to offer in this area. If the functional correctness of a security system cannot be guaranteed after an attack or mishap, it should at least be possible to ensure that the system 'fails safe' in that it always reverts to a *known* state which would be designed to contain the damage as far as possible. In practice, this would mean a state in which minimum (and preferably zero) information was disclosed to the attacker. This was one of the aims of the CSS.

The problem of *covert channels* has been briefly touched upon in section 4.2.2. and elsewhere. While integrity protocols can be devised to minimise this threat, the very mechanisms that can help to prevent this problem can be exploited to exacerbate it. In particular, the recent cryptographic concept of a *subliminal channel* [SEBE,1988] could be a difficult threat to deal with. This concept involves the notion of communication on 'two levels' via an encoding mechanism. There is the 'overt' contents of the message for all to see, but buried within the 'noise' is another *subliminal* communication channel which exploits the unused information bandwidth of the overt channel.

It is possible that artificial intelligence (AI) techniques could be employed, especially in the SMCs for intelligent supervision of the connections. This could allow potential problems to be predicted before they became serious and remedial action taken.

Another recent development is that of *zero knowledge*. This is a mechanism by which it is possible to exchange secrets with neither party being able to discover the secret of the other party until the exchange is complete. This eliminates the problem which can arise in the normal protocols where one party must transmit first. At this stage, it is possible for the other party to refuse to cooperate further, thus discovering the secret of the sender but not revealing anything in return. Some of the protocols used by the CSS require this two-way cooperation such as symmetric key exchange. The extension of the CSS protocols to use zero knowledge might provide an elegant solution to these problems.

7.4. The Future of Distributed Systems

The future of distributed systems is assured. The adoption of global information technology strategies is likely within the next twenty years and as the amount of potentially sensitive information carried increases (such as credit references, medical records and so forth) the need for security will be paramount. As security awareness grows among commercial users to the extent already prevalent among the financial and military communities the onus will be on network providers to make available a security system *which is of demonstrable validity*. Failure to do this is likely to result in severe under-utilisation of the benefits and resources of the information technology revolution.

ANNEX 1: Circuit diagram of CSS Hardware and source code for ROM firmware

The circuit diagram of the PC version of the CSS local hardware is given overleaf followed by the assembler source code for the ROM firmware.

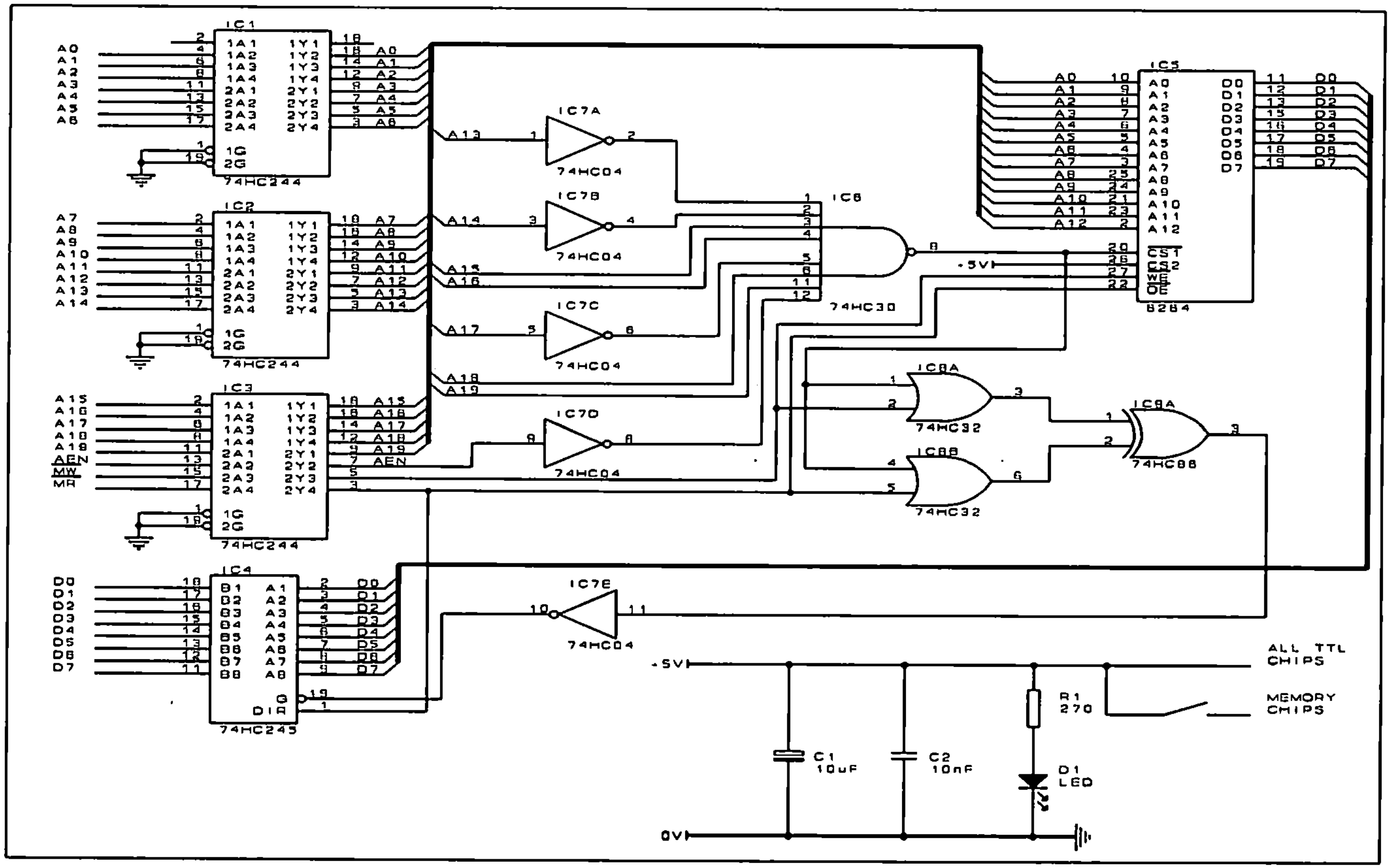


Figure A1.1. Circuit diagram of the PC version of the local CSS hardware

```
;       CSS SECURITY HARDWARE MODULE ASSEMBLER SOURCE
;       Simon J Shepherd

        CR      equ     0dh
        LF      equ     0ah

cseg    segment         byte    public  'code'          ; Start the code segment
        org             0                               ; Zero origin
        assume  cs:cseg,ds:cseg,es:cseg

start   equ     $

        db      055h                                    ; standard IBM ROM header
        db      0AAh
        db      03h                                     ; three 512 byte blocks

css_bios_init           proc    far

        jmp     code_start

table:  db      '0123456789ABCDEF'

msg_1:
        db      CR
        db      'NETWORK RESEARCH GROUP',CR,LF
        db      'Comprehensive Security System',CR,LF
        db      CR,LF
        db      'Version 2.0  for IBM PC/XT/AT/PS2 + BIOS COMPATIBLES',CR,LF
        db      CR,LF
        db      'Department of Electrical Engineering',CR,LF
        db      'University of Bradford',CR,LF
        db      'Bradford UK.',CR,LF
        db      '+44 (274) 384052',CR,LF
        db      CR,LF
        db      'This machine is fully controlled by the NRG Comprehensive',CR,LF
        db      'Security System.  The system will only be permitted to',CR,LF
        db      'boot from security device drivers in C:\CONFIG.SYS',CR,LF
        db      '*** BOOT FROM FLOPPY IS PROHIBITED ***',CR,LF
        db      CR,LF
        db      'Press a key to continue ...',CR,LF

code_start:
        mov     ax,ss                                   ; Get segment and offset of
        mov     es,ax                                   ; caller from stack and
        mov     si,sp                                   ; display
        lea     bx,cs:table
        mov     ah,0eh

        mov     al,'['
        int     10h
        mov     dx,es:[si+2]
        call    write_word
        mov     al,':'
        int     10h
        mov     dx,es:[si]
        call    write_word
        mov     al,']'
        int     10h

        mov     ax,cs                                   ; ES:BP points to message
        mov     es,ax

        mov     bp, offset msg_1
        mov     ax,1301h                                ; Write chr/attr TTY
        mov     bx,004fh                                ; Page 0  White on Red
        mov     cx,offset code_start - offset msg_1     ; Length of message
        mov     dx,0101h                                ; GotoXY (1,1)
        int     10h

        mov     ah,0                                    ; press a key...
        int     16h

        sub     ax,ax                                   ; address zero segment
        mov     es,ax                                   ; (vectors & ICA)

        mov     ax,word ptr es:[4*40h]                  ; get old int $40 vec offset
        mov     bx,es:[4*40h+2]                         ; & segment
        mov     word ptr es:[04f0h],ax                  ; save offset at 0000:04f0
        mov     es:[04f2h],bx                           ; save segment at 0000:04f2
        mov     ax,055AAh                               ; signature
        mov     word ptr es:[04f4h],ax                  ; store sig

        mov     ax,offset int_40                        ; load our int $40 handler address
        cli
        mov     word ptr es:[4*40h],ax
        mov     es:[4*40h+2],cs
        sti

        ret                                             ; far return back to BIOS POST

css_bios_init endp
```

```
handler_40 proc far

int_40:
          mov     ah,80h
          stc                                               ; fake a timeout error
          iret                                              ; back to BIOS bootstrap

handler_40 endp

write proc near

write_word:
          mov     al,dh
          call    write_hi_nybble
          mov     al,dh
          call    write_lo_nybble
          mov     al,dl
          call    write_hi_nybble
          mov     al,dl
          call    write_lo_nybble
          ret

write_hi_nybble:
          mov     cl,4
          shr     al,cl
          jmp     write_hex

write_lo_nybble:
          and     al,0fh

write_hex:
          xlat
          int     10h
          ret

write endp


IF        ($ - start) MOD 512
org       ($ - start) + 512 - (($ - start) MOD 512)
ENDIF

db        600 DUP (00h)

cseg      ends
end
```

ANNEX 2

A General Solution to Primality Testing and the Integer Factoring Problem

Reference: Bressoud, D.M. *Factorization and Primality Testing*, Springer Verlag, NY, 1989, ISBN 0 387 97040 1.

Groups

Definition: A *group*, G is a set together with a binary operation, say ∂, such that

1. The operation is *closed*. If x and y are in G then $x\partial y$ is also in G.
2. The operation is *associative*. If x, y and z are in G, then $(x\partial y)\partial z = x\partial(y\partial z)$.
3. G contains an identity, say e. For each x in G, $x\partial e = e\partial x = x$.
4. Each element of G has an *inverse*. If x is in G, then there is a y in G such that $x\partial y = y\partial x = e$.

The integers with addition form a group. Zero is the identity and $-x$ is the inverse of x. This group is called **Z**.

If n is any positive integer, the integers less than and relatively prime to n together with multiplication modulo n form a group. This group is called $U(\mathbf{Z}/n\mathbf{Z})$.

The *order* of a group G, denoted by $|G|$, is the number of elements in G.

Given a group G with binary operation ∂, identity e, and an element x in G, the *powers* of x in G are defined as follows:

$x\#{-1}$	=	the inverse of x,
$x\#0$	=	e,
$x\#1$	=	x,
$x\#2$	=	$x\partial x$,

$x\#3 = x\partial x\partial x$, and so on.

and in general

$$x\#i = x\partial(x\#(i\text{-}1)) = (x\#\text{-}1)\partial(x\#(i+1)).$$

The *order* of an element x in G is the smallest positive integer i such that

$$e = x\#i$$

If the group G has finite order, then every x in G has a finite order, and if x is an element of G then the order of x divides the order of G.

G is a *group modulo n* if its elements are vectors of residues modulo n and its binary operation ∂ is defined in terms of arithmetic modulo n. If d is any divisor of n, then the restricted group modulo d, denoted $G|d$, is the group derived from G by reducing each coordinate modulo d.

A General Approach to Primality Tests

Let n be a candidate for primality and assume we have a group G whose elements are a subset of the residues modulo n or some subset of vectors of residues modulo n. Further assume that the possible orders of elements of G depend on the factorization of n in such a way that an element of order m can exist if and only if n is prime. If we know the factorization of m, we can prove that an element x in G has order m if we can verify that

$$x\#m = e \qquad \text{and} \qquad x\#(m/p) \neq e$$

for every prime p dividing m. Formalising this in a theorem,

Theorem: Let n be a suspected prime, and assume that there exists a group modulo n, say G. Let $G|d$ be the restricted group modulo d and let e be the identity in G. If we can find an element x in G and an integer m satisfying the following conditions, then n is definitely prime:

1. The integer m is larger than the order of $G|q$ would be for any prime q

dividing n and less than the square root of n,

2. $x\#m = e$,

3. For each prime p dividing m, some coordinate of $x\#(m/p) - e$ is relatively prime to n.

A General Approach to Factorization

The following is a general solution to the factoring problem, and represents the underlying theory of most of the factoring algorithms in existence, including:

Fermat's Method	$gcd\ (x - y,\ n)$
Euler's Method	$gcd\ (ad - bc, n)$
Shank's SQUFOF	$gcd\ (A_{n-1} \pm R, n)$
Morrison & Brillhart's Method	$gcd\ (x + y, n)$
Pollard's Rho	$gcd\ (x_{2i} - x_i, n)$
Pollard's p-1	$gcd\ (b^{k!} - 1, n)$
William's p+1	$gcd\ (b^{k!} - 1, n)$
Legendre's Method	$gcd\ (x - y, n)$
Lucas Sequences	$gcd\ (V_{k!} - 2, n)$
Kraitchik's Method	$gcd\ (x - y, n)$
Pomerance's Quadratic Sieve	$gcd\ (x - y, n)$
Lenstra's Elliptic Curve Algorithm	$gcd\ (x - y, n)$
Pollard's Number Field Sieve	$gcd\ (x - y, n)$

Let n be an integer known to be composite, and p be an unknown prime divisor of n. Let G be a group modulo n and G$|p$ the restricted group modulo p. If the order of G$|p$ is considerably less than the order of G, then we can hope to find an element x in G and an integer k such that $x\#(k!)$ is *not* the identity in G, but the corresponding computation in G$|p$ does yield the identity of G$|p$. This means that there is at least one coordinate of $x\#(k!) - e$ which is *not* divisible by n, but *all* of the coordinates are divisible by p. Taking the greatest common divisor of n and the coordinate which is *not* divisible by n will yield a non-trivial divisor of n.

Of course, if n is prime, the above approach will run for ever and produce only inconclusive results. Even if n is composite, there is no way of knowing *a priori* that the order of $G|p$ will divide $k!$ for some prime p dividing n. We observe in addition that this approach also requires an efficient means of calculating at least one coordinate of $x\#(k!)$.

Summing up this idea in a theorem,

Theorem: Let n be a composite integer and let G be a group modulo n. Let p be a prime dividing n and let $G|p$ be the restricted group modulo p. If the order of $G|p$ divides $k!$, then p divides each coordinate of $x\#(k!) - e$. If n does not divide the i^{th} coordinate of $x\#(k!) - e$, then the greatest common divisor of n and the i^{th} coordinate of $x\#(k!) - e$ is a non-trivial divisor of n.

Factorization Times

The following table illustrates the expected time to factor various sizes of integers of no special form using a CRAY II supercomputer.

DIGITS	**BITS**	**TIME**
15	50	3 seconds
50	166	1 minute
75	250	10 hours
100	332	2 months
176	585	1 million years
200	664	15 million years
255	847	15 billion years ($15 * 10^9$)
300	996	15 trillion years ($15 * 10^{12}$)
500	1660	$15 * 10^{20}$ centuries (150 billion trillion years)

The current estimate for the present age of the universe is about 15 billion years, which corresponds roughly to the time required to factor a 255 digit number. This is the order of the key size used by the encryption mechanisms in the CSS.

Some keys generated by the CSS QRC Key Generator

60 digit key

P_k = 21047754979673754543478202726855879255167357908757
1155514269

S_{k1} = 79115453021138497575239

S_{k2} = 26603848143358416653158919663907957 71

100 digit keys

P_k = 10001509679053340702344207115108909674215089803237
27498472532019952554144341496528612572834844254 41

S_{k1} = 44306346841216415011486282093426358611

S_{k2} = 22573537183953378737679652181686361082356367391 61
604315473531

P_k = 64563289550257326009017688393606227159530449075 09
95548918468878485191074142090618049924015368028 77

S_{k1} = 19407102709770001680177215003984139911

S_{k2} = 33267866160029449002234550616391746934600467233 53
447648788507

180 digit key

P_k = 56231048169461842154453183945054559436655899401113
92774549637493001334874736683792770298866508348023
25794856280342249679351025648365697483629682773535
8628475081193516075996455682 09

S_{k1} = 68795365326949132093237486000865997241233651371299
893723575958300103

S_{k2} = 81736680810145383549986337251603531293116722637991

86000534575123232552160576135322389400577983143044
622607579303

200 digit keys

P_k = 27529629391493083271949844202075242699600701123227
19856290275025195586459675229537846318441585847638
60957247936200705406098926554618604902759111387772
49440403691706709326706330050993014191539473777629

S_{k1} = 30932250113805811778757012199202741885275698136910
84049497414560901 11864903

S_{k2} = 88999763322119083293455152349074573302883387702693
95854819275900267497996880495580175697245767643118
9418179437137016813715643

P_k = 11062149228898998505918983166254647000543307822872
581475111513818118711542735206718616564274389444099
19541197394923874561847893631086436409005332464949
72630062285236189195544751841124291941686415754801

S_{k1} = 19982264781855869999789558687431224364527302406360
7834266413527717027085203

S_{k2} = 55359837083850271638465552841068131727326890997939
69993717549540671364527686272299297623488771851934
9347502953926899453655467

250 digit key

P_k = 25493167138607526311988918211048849847392554872720
82139428191050897557320921554113363060166734786595
79605465598321908644578575376161944823317657734488
63430123179494554019455701247838406999529271731964
98080509599473991501838989720165350503180262678193

S_{k1} = 29902544272176037747513316548240492723898386928364
22065757375502108156874723067644718147202318879502

7994302084134592928984147

S_{k2} = 852541740480879933611475989640958964078825412310532866720568265599397021616950765074474749915385716716449120328502323674502 19

255 digit key

P_k = 359970338939925441574488874379804331081366604838992192553576696241151505436650192768815644488491451759218541270255525536998667250764838256113849450165498790537916365661882064624268194766684533194662795903400338955637056774328506233365873486903122418641 49

S_{k1} = 5056764698875685081313573023147875293020748892249993898014962563919355113058138816509663294861008964362746920973790579375683483

S_{k2} = 71185898568695674001871629425665213707472058103218570416998331486672437918367997507528690088555196304499773706896777312168139 03

ANNEX 3: SOURCE CODE LISTINGS

The complete source code listings of the Comprehensive Security System are supplied on floppy disk, bound within the *master copy* of this thesis.

ANNEX 4: INTRODUCTION TO PREDICATE CALCULUS

Simple ideas, the truth of which can be captured in *true* or *false*, are called *propositions*, and are denoted by letters. For example, the proposition P,

P : "Esmerelda is a duck"

is either true or false. The formalism allows the capture of truth about propositions by providing some components of the formal language which map onto properties of the object. These are called *predicates*. The property of 'being a duck' can be captured, for example, by the predicate *duck (x)*, where *x* is a *free variable*, and can be filled by the names of suitable objects to create propositions. We can write, for example, *duck(Fred)* which will be true if Fred is a duck but false if Fred is a dog. The act of assigning an instance to a predicate to form a propositions is called *instantiation*. The predicate itself is not said to be *truth valued*, in other words, it cannot be ascribed the values true or false. It is only when the predicate is instantiated with an object that the resulting proposition can be truth valued. It is important to realise that the instantiation of general objects can lead to difficult philosophical problems. For instance, *duck(hope)* is difficult, because while the instantiation is valid, the 'duckness of hope' is a difficult concept to grasp! The predicate *duck(x)* is called a *unary* predicate, since there is only one place for an object to be instantiated. It is quite possible, however, to have *n*-ary place predicates, such as *father(x,y)* where someone can clearly be the father of more than one other person.

Applying the formalism, the rules of inference discussed previously can be applied to predicates to obtain inferences which are new. For example, starting with the propositions

'Esmerelda is a duck'
'Ducks like water'

we can infer the result that *'Esmerelda likes water'*. It is very important not to draw false inferences, however. If the two propositions were

'Sam is cold'

'Sam is wet'

the formalism does *not* infer that Sam is cold *because* he is wet. The concept of *consequence* is the result of far higher levels of abstraction of which the human mind is capable.

We now extent the above formalism to include the notion of *quantification*, and now predicates can give rise to propositions in two ways. First, free variables can be instantiated with the names of objects as described above. Second, quantification can be used to effect the instantiation process itself. Quantification introduces two additional symbols to the formalism, $\forall$ and $\exists$. These symbols are used to capture the ideas of *universal* and *existential* qualification respectively. Universal quantification allows propositions of the form "every object has this property" or "for all objects of this form". By contrast, the existential quantifier captures the notion of 'someness', "there exists exactly one or more". When a quantifier is applied to a predicate with a free variable(s), the variable is said to be *bound* by the quantifier because it can no longer be freely instantiated.

To give examples of these quantifications, the *assertion*

$$\forall x \bullet duck(x)$$

captures the (false) assertion that "every object is a duck". By contrast, the assertion

$$\exists x \bullet duck(x)$$

captures the (probably true) assertion that "there exists at least one duck".

Finally, we are in a position to use the formalism to discover inferences about real situations. Let us explore the ways of capturing the notion "ducks like the pond", where *pond* is an identifiable object in the *domain of interest*. We choose *duck(x)* to denote the assertion that some object is a duck, *pond* to represent the pond, and *likes(x,y)* to denote x likes y. We now carefully abstract the semantics of what we are trying to say. The original notion could be rephrased "whatever value we choose for x, if x is a duck than x likes the pond". The IF..THEN idea has already been encountered, and is reflected by the symbol $\Rightarrow$ in the formalism. (Note that in Chapter 6 where a modified form of this logic is used, the $\Rightarrow$ symbol is given a completely different meaning). This sort of reasoning leads to a sentence of the form

$$\forall x \bullet (duck(x) \Rightarrow likes(x,pond))$$

Clearly, if we do not instantiate this with $x = duck$ then the sentence will (probably) not be true. We need to try another approach. Let us try the phrase "there exists a duck that likes the pond". The major difference between this and the previous idea is that the notion that is now being expressed is that there is in our domain of interest *at least one* duck. Note very carefully that we are not thinking along the lines of "if there is a duck then...", but along the lines of "there is a duck and...". Within the formalism we use the symbol $\wedge$ to represent the logical 'and' concept. With the benefit of this insight we construct the sentence

$$\exists x \bullet (duck(x) \wedge likes(x,pond))$$

This states that within our domain of interest there exists (at least one) duck, and that it likes the pond. But if we want to learn from this observation and obtain the possible (new) piece of knowledge that "ducks like water", it is necessary to progress the argument. We now need to say that not only do ducks like the pond, but that they also like any object that has the property of being made of water. (Note that when the *pond* was introduced, its existence was assumed, but now we cannot be sure if there exist any objects which are wet). Let the predicate $water(y)$ denote that y has the property of being wet, and we wish to express the idea that "if x is a duck and y is a wet object, then x likes y", and that this be true for whatever objects we choose for x and y. We can write

$$\forall x \bullet \forall y \bullet ((duck(x) \wedge water(y)) \Rightarrow likes(x,y))$$

This sentence will be true even if there are no ducks and no water.

Finally, we can use the formalism to eliminate ambiguities which arise as a consequence of natural language. The natural language sentence "there is a duck who likes every wet object" can be interpreted two ways. Firstly, it could mean that is one specific, water-loving duck who likes every wet object. Secondly, it could mean that for every wet object, there is a duck somewhere that likes it. Ambiguities such as these cannot arise in the formalism, because the two formal sentences expressing the two possibilities would be quite different.

The ideas discussed above can be extended much further to create large bodies of knowledge such as abstract set theory, theories of functions and specialised algebras. These can then be applied to software constructs to yield a representation of the software in a rigorous way.

GLOSSARY OF TERMS AND ABBREVIATIONS

ACSE	Association control service element
AI	Artificial intelligence
API	Application program interface
ASE	Application service element
CCITT	International Telegraph & Telephone Consultative Committee
CRC	Cyclic redundancy check
CSS	Comprehensive Security System
DES	National Bureau of Standards Data Encryption Standard
DMM	Data manipulation mechanism
DTI	Department of Trade and Industry
DU	Data unit
EBCDIC	Extended binary coded decimal interchange code
ECM	Elliptic curve method (integer factoring algorithm)
ECMA	European Computer Manufacturers Association
EEA	External environment agent
EEC	European Economic Community
FADU	File access data unit
FEAL	Fast encryption algorithm
FPDU	FTAM protocol data unit
FTAM	File Transfer, Access and Management
IDCA	Inter-domain communication agent
IO	Input/output
ISO	International Standards Organisation
ISORM	ISO Reference Model (refers to the OSI model)
JANET	British Universities Joint Academic Network
LAN	Local area network
MA	Monitoring agent
MHS	Message Handling System
MOTIS	Message Oriented Text Interchange System
MPQS	Multiple polynomial quadratic sieve (integer factoring algorithm)

NFS	Number field sieve (integer factoring algorithm)
NIST	(American) National Institute of Standards
NPDU	Network protocol data unit
ODP	Open distributed processing
OSI	Open Systems Interconnection
OSIRM	OSI Reference Model
PC	Personal Computer (IBM compatible)
PDU	Protocol data unit
PGM	Protocol generation mechanism
PSAP	Presentation service access point
QOS	Quality of service
RSA	Rivest, Shamir & Adleman (Cryptosystem)
SAA	Security administrator agent
SAP	Service access point
SASE	Specific application service element
SDU	Service data unit
SMA	Security mechanism agent
SMC	Security Management Centre
SMIB	Security management information base
SMIBA	SMIB agent
SS	Session (layer) service
SPDU	Session protocol data unit
SSA	Security services agent
SSDU	Session service data unit
TS	Transport service
TPDU	Transport protocol data unit
TSDU	Transport service data unit
UA	User Agent
WAN	Wide area network

REFERENCES AND BIBLIOGRAPHY

[ADLE,1991] Adleman, L. M. *Factoring Numbers using Singular Integers*, Proceedings of 23rd Annual ACM Symposium on the Theory of Computing (STOC), 1991, pp 64-71.

[AMS,1983] DeMilo, R.A. *et al.* *Applied Cryptology, Cryptographic Protocols and Computer Security Models*, Proceedings of Symposia in Applied Mathematics, American Mathematical Society, Providence, RI, ISBN 0 8218 0041 8.

[BELL,1973] Bell, D.E. and La Padula, L.J. *Secure Computer Systems : Mathematical Foundation and Model*, M74-244. Bedford, MA, 1973, Mitre Corp, Report NTIS AD-771543.

[BLAT,1987] Blatchford, C.W. *Security in Distributed Information Systems: Needs, Problems and Solutions*, ICL Technical Journal, 1987.

[BLUM,1986] Blum, L., Blum, M. & Shub, M. *A Simple Unpredictable Pseudo-random Number Generator*, SIAM Journal on Computing, Vol 15, 2, 1986, 364-383.

[BLUM,1989] Blum, M. & Goldwasser, S. Proc. CRYPTO 89, Santa Barbara, CA, not yet published.

[BOEB,1985] Boebert, W.E., Kain, R.Y., Young, W.D. and Hansohn, S.A. *Secure ADA Target: Issues, System Design and Verification*, Proceedings of the 1985 Symposium on Security and Privacy, Silver Spring MD, IEEE Computer Society, 176-183.

[BRAS,1987] Brassard, G. *Cryptology in Academia: A Ten Year Retrospective*, IEEE Compcon Conference, San Francisco, Feb 1987, 222-226.

[BRAS,1988] Brassard, G. *Modern Cryptology*, Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1988, 35-39.

[BURR,1988] Burrows, M., Abadi, M. and Needham, R. *Authentication: A practical Study in Belief and Action*, Proc. 2nd Conf. on Theoretical Aspects of Reasoning About Knowledge, Asilomar, Feb 1988.

[CARO,1988] Caron, T.R. and Silverman, R.D. *Parallel Implementation of the Quadratic Sieve*, Journal of Supercomputing, 1, 1988, 273-290.

[CCIT,1987a] *The Directory - Authentication Framework*, CCITT Draft Recommendation X.509, Version 6, Geneva, June 1987.

[CCIT,1987b] *Message Handling, System Service and Overview*, CCITT Draft Recommendation X.400, Version 4, November 1987.

[CLAR,1987] Clarke, D.C. and Wilson, R.W. *A Comparison of Commercial and Military Computer Security Policies*, Proceedings of the 1987 Symposium on Security and Privacy, Washington DC, 184-195.

[COOP,1988] *The Security of Network Systems*, Coopers & Lybrand report for the Commission of the European Communities, 5 volumes, ISBN 0 950 51271 0.

[DAVI,1985] Davies, J.A., Holdridge, D.B. and Simmons, G. J. *Status Report on Factoring (at the Sandia National Laboratories)*, Advances in Cryptology, Lecture Notes in Computer Science #209, Springer, Berlin, 1985.

[DENN,1983] Denning, D.E. *Cryptography and Data Security*, Addison-Wesley, Reading MA, 1983.

1976
[DIFF,~~1984]~~ Diffie, W. and Hellman, M.E. *New Directions in Cryptography*, IEEE Transactions on Information Theory, Vol IT-22, 1976, 644-654.

[DIXO,1984] Dixon, J.D. *Factorization and Primality Tests*, American Mathematical Monthly, 91, 1984, 333-352.

[DOD,1985] *US Department of Defense - DoD Trusted System Evaluation Criteria (The Orange Book)*, DOD 5200.28-STD, Washington DC., 1985, US Government Printing Office number 008-000-00461-7.

[DTI,1989] *Evaluation and Certification Manual*, V23, Version 3.0, (5 volumes) The Department of Trade and Industry.

[ECM,1987a] *Framework for Distributed Office Application*, European Computer Manufacturers Association, Document TR/42, July 1987.

[ECM,1987b] *Security in Open Systems*, European Computer Manufacturers Association, Document TR/46, December 1987.

[ENSL,1978] Enslow, P.H. *What is a Distributed Data Processing System?*, Computer, **11**, Jan 1978, 13-21.

[ESPR,1988] *Protection of Electronic Information in an open Network Environment*, Thorn-EMI Consortium, ESPRIT proposal, Feb 1988.

[GASS,1988] Gasser, M. *Building a Secure Computer System*, Van Nostrand Reinhold, New York, NY, 1988, ISBN 0 442 23022 2.

[GOGU,1982] Goguen, J.A. and Meseguer, J. *Security Policy and Security Models*, Proceedings of the 1982 Symposium on Security and Privacy, Silver Spring MD, IEEE Computer Society, 11-20.

[HALS,1988] Halsall, F. *Data Communications, Computer Networks and OSI*, Addison Wesley, 1988, ISBN 0 201 18244 0.

[HANT,1976] Hantler, S.L. and King, J.C. *An Introduction to Proving the Correctness of Programs*, ACM Computing Surveys, September 1976, 331-353.

[HARR,1976] Harrison, M.A., Ruzzo, W.L. and Ullman, J.D. *Protection in Operating Systems*, Communications of the ACM, 1976, Vol 19, 8, 461-471.

[HOAR,1969] Hoare, C.A. *An Axiomatic Basis of Computer Programming*, Communications of the ACM, 1969, 12, 576-583.

[HYLA,1988] Hyland, P. and O'Connell, J. *Model Checking in Temporal Logic*, Hewlett Packard Internal Report, Feb 1988.

[ISO,7498] International Standard ISO 7498, *Open Systems Interconnection, Basic Reference Model.*

[ISO,7498-2] ISO 7498-2: *Open Systems Interconnection : Basic Reference Model - Security Architecture*, ISO, Feb 1989.

[ISO,8571] ISO 871: *Information Processing Systems - OSI- File Transfer, Access and Management*, ISO.

[ISO,JTC1] ISO/IEC JTC1: *ISO Technical Report on Security in Open Systems*, ISO, June 1988.

[ISO,SC21] ISO/IEC JTC1/SC21: *Working Draft : Access Control Framework*, ISO, Dec 1988.

[ISO,TC97] ISO TC97/SC20/WG3 *Link, Network, Transport and Presentation Layer Encryption*, ISO, 1985-87.

[ITSE,1990] ITSEC *Information Technology Security Evaluation Criteria*, Harmonised Criteria of France, Germany, the Netherlands and the United Kingdom, 2 May 1990, draft copy.

[JOHN,1988] Johnson, D. and Thayer, J. *Starting Security Requirements with Tolerable Sets*, ACM Transactions on Computer Systems, Vol 6, 3, Aug 1988, 284-295.

[KNUT,1981] Knuth, D. *The Art of Computer Programming, Vol 2 : Semi-Numerical Algorithms*, Addison-Wesley, Reading, MA, 1981, 378-380.

[KOBL,1987] Koblitz, N. *A Course in Number Theory and Cryptography*, Springer Verlag, New York, NY, 1987, ISBN 0 387 96576 9.

[LAND,1981] Landwehr. C.E. *Formal Models for Computer Security*, Computing Surveys, 13, No 3, 247-278. Reprinted in Advances in Computer System Security, 2, R. Turn (ed), 76-107, Dedham MA, 1981, Artech House Publications.

[LENS,1986] Lenstra, H. *Factoring Integers with Elliptic Curves*, Mathematisch Instituut, University of Amsterdam, Report 86-16, July 1986.

[MART,1982] Martin, J. *Program Design which is Provably Correct*, Savant Research Studies report, Savant, December 1982.

[MATY,1984] Matyas, S.M. *Digital Signature with the Data Encryption Standard*, Securicom 84, 2nd Worldwide Congress on Computer and Communication Security and Protection, Cannes, France, 29 Feb - 2 Mar 1984, 149-159.

[McMO,1989] McMorran, M.A. and Nicholls, J.E. *Z User Manual*, Draft Version 0.65, Internal Document, IBM (United Kingdom) Laboratories, Sheridan House, Winchester, UK, private communication.

[MERK,1978] Merkle, R.C. *Secure Communications over Insecure Channels*, Communications of the ACM, 21, 1978, 294-299.

[MILL,1982] Miller, G.L. *Riemann's Hypothesis and Tests for Primality*, Proceedings of the 7th ACM Symposium on the Theory of Computing, 1982.

[MILL,1984] Millen, J.K. and Cerniglia, C.M. *Computer Security Models*, MTR-9531, Bedford MA, 1984, Mitre Corp, Report NTIS AD-A166920.

[MORI,1989] Morita, H. *A Fast Modular-multiplication Algorithm based on a Higher Radix*, Proc. CRYPTO 1989, University of California, Santa Barbara, USA, 387-399.

[MUFT,1988] Muftic, S. *An Integrated Network Security Mechanism*, Proc. EUTECO 88, CEC Conf. Vienna, April 1988.

[MUFT,1989] Muftic, S. (editor) *Security Mechanisms for Computer Networks, Vol 1*, Ellis Horwood, Chichester, England, 1989, ISBN 0-7458-0613-9.

[MUFT,1990] Muftic, S. *Specification of Security Requirements for Open Distributed Environment*, private communication.

[MUFT,1992] Muftic, S. (editor) *Security Mechanisms for Computer Networks, Vol 2*, Ellis Horwood, Chichester, England, 1992.

[NEED,1978] Needham, R.M. and Schroeder, M.D. *Using Encryption for Authentication in Large Networks of Computers*, CACM, Vol 21, 12, Dec 1978, 993-999.

[OTWA,1987] Otway, D. and Rees, O. *Efficient and Timely Mutual Authentication*, Operating Systems Review, Vol 21, 1, Jan 1987, 8-10.

[PATE,1988] Patel, A. and Law Min, F. *Study of Security Measures within the OSI Framework*, University College Dublin, private communication.

[PIER,1988] Pierson, L.G. and Witzke, E.L. *A Security Methodology for Computer Networks*, AT&T Technical Journal, May/June 1988.

[POLI,1988] Polis, R.I. *European Needs and Attitudes towards Information Security*, Cryptologia, Vol XII, 4, Feb 1988, 234-239.

[POLL,1989] Pollard, J.M. *The Number Field Sieve*, private communication.

[POME,1984] Pomerance, C. *The Quadratic Sieve Factoring Algorithm*, Proc. EUROCRYPT 84: Advances in Cryptology, 169-182.

[RABI,1980] Rabin, M.O. *Probabilistic Algorithm for Primality Testing*, Journal of Number Theory, **12**, 128-138.

[RAMA,1990] Ramaswamy, R. *A Key Management Algorithm for Secure Communication in the OSI Architecture*, Computers and Security, **9**, Elsevier Science Publishers, 1990, 77-84.

[RAYM,1988] Raymond, M. *Network Security Problem for the EEC*, Communications, June 1988, 11-13.

[RIES,1987] Riesel, H. *Prime Numbers and Computer Methods for Factorisation*, Progress In Mathematics, **57**, Birkhauser, Boston, 2nd ed.

[RIVE,1978] Rivest, R.L., Shamir, A. & Adleman, L. *A Method for Obtaining Digital Signatures and Public Key Cryptosystems*, Communications of the ACM, **21**, 1978, 120-126.

[RUSH,1986] Rushby, J. *Introduction to Dependable Computing for Communications*, Communications of the ACM, **29**, 1986.

[RUTL,1986] Rutledge, L. and Hoffman, L. *A Survey of Issues in Computer Network Security*, Computers & Security, **5**, Dec 1986.

[SAN,1988a] Sanders, P.W. *Security Management Centre Concept*, CEC COST 11 *ter* Discussion Paper, London, Jan 1988, private communication.

[SAN,1988b] Sanders, P.W. *Extended Security Architecture*, CEC COST 11 *ter* Discussion Paper, London, Oct 1988, private communication.

[SEBE,1988] Seberry, J. and Pieprzyk, J. *Cryptography*, Prentice Hall, England, 1988.

[SHEP,1990a] Shepherd, S.J., Sanders, P.W. and Stockel, C.T. *The Quadratic Residue Cipher and some Notes on Implementation*, accepted by Cryptologia in March 1992, to appear.

[SHEP,1990b] Shepherd, S.J. *A Preliminary Investigation into Continuous Authentication by Analysis of Keyboard Typing Characteristics*, to appear in Electronics Letters.

[SHEP,1990c] Shepherd, S.J., Sanders, P.W. and Patel, A. *A Comprehensive Security System - the Concepts, Agents and Protocols*, Computers & Security, 9, 1990, 631-643.

[SIDE,1988] Sidey, G. *UK Government Policy on OSI*, Computer Standards and Interfaces, Vol 7, Part 1-2, 1988, ISSN 0920-5489, 89-93.

[SILV,1987] Silverman, R.D. *The Multiple Polynomial Quadratic Sieve*, Mathematics of Computation, Vol 48, 177, Jan 1987, 329-339.

[SMIT,1989] Smith, P. *Security Standards and Literature Guide*, OSITOP Working Group 5 Internal Document, private communication.

[SNAR,1983] Snare, J.L. *An Introduction to the CCITT Recommendation X.25*, Telecomm. Journal of Australia, 33, 1983, 113-124.

[SPAF,1988] Spafford, E.H. *The Internet Worm Program : An Analysis*, CDS-TR-823, November 1988, Purdue University, Indiana, USA.

[STOL,1988] Stoll, C. *Stalking the Wily Hacker*, Communications of the ACM, Vol 31, 5, May 1988.

[SUMM,1987] Summers, R.C. *An Overview of Computer Security*, IBM Systems Journal, 23, 1987, 309-325.

[TANE,1981] Tanenbaum, A.S. *Computer Networks*, Prentice Hall Inc, Englewood Cliffs, NJ, 1981, ISBN 0-13-164699-0.

[TURI,1936] Turing, A. *On the Computable Numbers, with an Application to the Entschiedungsproblem*, Proceedings of the London Mathematical Society, Ser 2-42, 1936, 230-265.

[VARA,1988] Varadharajan, V. *Analysing Some Formal Properties of a Security Protocol using State Machine Techniques*, Hewlett Packard Internal Paper, Jul 1988.

[VARA,1989] Varadharajan, V. *Verification of a Network Security Protocol*, Computers & Security, Vol 8, 8, Dec 1989.

[VOYD,1983] Voydock, V. and Kent, S. *Security Mechanisms in High-Level Network Protocols*, Computing Surveys, 15, June 1983.

[WEIN,1987] Weingart, S.H. *Physical Security for the ABYSS System*, Proceedings of the 1987 IEEE Symposium on Security and Privacy, April 27-29, Oakland, CA.

[WEST,1978] West, C.H. *General Technique for Communications Protocol Validation*, IBM Research Journal, 22, July 1978, 393-404.

[YAO,1982] Yao, A. *Theory and Application of Trapdoor Functions*, IEEE 23rd Symposium on the Foundation of Computer Science, 80-91.

ADDENDUM: LIST OF PUBLICATIONS

During the course of this the research, the author has been primary contributor to the following publications

[1] *A Comprehensive Security Service - Functional Specification*, IBM Internal Document, IBM (United Kingdom Laboratories), Hursley Park, Winchester, UK, May 1989.

[2] *A Distributed Security Architecture for Large Scale Systems*, presented to the International Federation of Information Processing, International Workshop on Distributed Systems Operations and Management, Berlin, October 22-23, 1990.

[3]* *A Comprehensive Security System - the Concepts, Agents and Protocols*, with Sanders, P.W. and Patel, A. **Refereed article,** Computers and Security, Vol 9, 7, pp 631-643, Elsevier Science Publishers, November 1990.

[4] *Security Mechanisms for Computer Networks, Volume 2: Chapter 3: Extended OSI Security Architecture*, Muftic, S. (Ed), Ellis Horwood, Chichester, England, 1992, to appear.

[5]* *The Quadratic Residue Cipher and some Notes on Implementation*, accepted by *Cryptologia* in March 1992, to appear.

* Copies bound within this Addendum to the thesis.

Computers & Security, 9 (1990) 631–643

A Comprehensive Security System— the Concepts, Agents and Protocols

S. J. Shepherd[1], P. W. Sanders[1] and A. Patel[2]

[1]*Network Research Group, Department of Communication Engineering, Polytechnic South West, Plymouth, U.K.*
[2]*Department of Computer Science, University College Dublin, Dublin, Ireland*

This paper presents an overview of a comprehensive security architecture for use within, and as a value-added enhancement to, the ISO Open System Interconnection model. The system is arranged basically as an application layer service but can allow all of the ISO-recommended security facilities to be provided at any layer of the model. It is suitable as an "add-on" service to existing arrangements or can be fully integrated into new applications. For large-scale, distributed processing operations, a network of "security management centres" is suggested, that can help to ensure that system misuse is minimized, and that flexible operations are provided in an efficient manner.

Keywords: Security, Protocols, Computer networks, Security management, Policy, Agents.

1. Introduction

Open distributed processing (ODP) is the conceptual framework within which systems of diverse application and location can interact freely if required. Because there may be many different components, operations, resources and entities involved in such an arrangement, a network constructed within this framework presents a very convenient target for various attacks and illegal operations, which means that protection of the system resources and assets is becoming an increasingly important factor in network design.

The comprehensive security system (CSS) to be described involves the provision of security services for use at, and for the transfer of data between, remote end user entities. Within the ISO model, there are potentially many different services and applications which will benefit from a value-added security system, but in general, each requiring a different combination or sequence of security functions.

Systems currently exist which have made some attempt to implement security measures. In many cases, the most effective are those which were conceived from the outset to offer security as a prime function, and are typical of those used by governments, military and financial institutions.

The majority of other communications systems which exist, however, were not originally conceived with the security function in mind, and make no provision for it other than allowing the execution of specific applications which have security measures built into their facilities on an individual basis. A typical example is the JANET academic network, which does not provide encryption facilities, but may be used to send encrypted messages if the appropriate mechanisms are provided by individual system users. A secure e-mail facility using the RSA algorithm has been implemented in the Polytechnic and is used for secure correspondence with collaborators at home and abroad. The disadvantage of this approach is that it is very difficult to assess the overall strength of such a system, where security is provided on an individual "ad hoc" basis, owing to the absence of a formal architecture capable of rigorous analysis.

The concept of a CSS, which can be retrospectively added to an existing data processing system as a value-added function provider, and the integrity of which can be demonstrated by formal models and methods of logical analysis, is very attractive to owners and managers of large, existing communication networks. New applications can be written to utilize the security functions on offer, and existing applications can be modified or updated to use the system, but may involve substantial low level programming.

2. The Security Function

Generally, security refers to a complex of measures, which may be broadly classified into [1]

• procedural; (*e.g.* selecting trustworthy personnel, changing passwords regularly, etc.)

• logical; (*e.g.* access controls and cryptography)

• physical; (*e.g.* vaults and doorlocks, screening against emanation of interpretable emissions, etc.)

which are aimed at the

• prevention;
• detection and indication;
• correction;

of certain kinds of system misuse, accidental and deliberate.

Security not only addresses attacks and threats external to the system, but internal attacks from known user entities. If guarantees of authentication can be provided, it is possible to devise a system in which all user entities are subject to strict access control, thus minimizing the internal threat. Of course, it is virtually impossible to stop a user passing information to an attacker directly, so user trust and strong enforcement procedures are also required.

Only authorized users can obtain/provide information which will help to eliminate, as far as possible, misuse of the system, such as eavesdropping on confidential data, abuse of resources, fraudulent activity, forgery of messages etc. The recommended range of services which a security system could provide is comprehensively addressed in ref. [2].

3. Common Security Measures

Currently, if a particular application requires security services, these are generally constructed by hardware/software means into the application itself from conception. In a system where there are several normal (insecure) applications, and one or two secure services, this approach is quite satisfactory.

By contrast, in a system where there are many possible applications, as with ODP, a large number of which may require security facilities, it is clearly wasteful for each application to provide a complete set of security services for its own private use, when a majority subset of the services could be common to most, if not all, applications.

A system was considered which attempted to overcome this problem of duplicated services, by seeking to intercept all input and output (both data and control) to and from applications, and to impose security functions upon the application by redirection of the data via a security system kernel.

In principle, all software application packages should be written to standardized specifications. For example, application layer entities within the ISO OSI seven-layer model should conform to the ISO reference model [2]. On investigation, however, this is far from the case in practice. Had all existing software adhered strictly to specifications, it may have been possible (albeit very tedius) to implement such an I/O redirection system, which could cope with the widely differing interface, data and control requirements of all the various applications. Since the majority of software applications are written as an amorphous entity, with no obvious interface standards, the concept was discarded as impractical. Even within the context of a local area network of personal computers, running MS DOS for example, the amount of operating system interrupt handling to account for all DOS file I/O alone, proves to be an extremely difficult task.

The concept of providing a security system which is independent of, but available to, specific applications on request, is therefore only possible if the applications themselves are modified to include a standardized application program interface (API). The requests for security services, control and data information and any other data must flow in a rigidly defined manner across the interface which shall enable analysis of the data flow protocols for formal demonstration of the strength of the system to be made.

4. Conceptual model of a comprehensive security system

4.1 Security Policy

Within an ODP environment, involving the transfer of information between remote end-user systems, the provision of a generic security function can be conceived in terms of a security policy, rigorously enforced upon those entities who are subject to that policy. The security policy represents the overall set of measures adopted to fulfil the desired security function and covers every aspect of the business of implementing an effective security system. It will involve:

(1) Provision of physical, hardware and software security mechanisms, such as locked and guarded buildings, protected terminals, encryption and so on;

(2) Definition of protocols for all data transfers within the system, either embedded within existing OSI protocols, or interfaced to them;

(3) An enforcement of the fundamental principles of access control, user identification/authentication;

(4) Provision for effective system resource protection and optimization of use. This includes such measures as integrity of resources, confidentiality of use of resources, assurance of service, accountability of usage of resources, audit trail etc.;

(5) Provision for monitoring, logging and analysis of the security system at all times, for both optimization of system resources, and detection of possible subversive activity.

The security policy is formulated and dictated by an authority, which is ultimately responsible for the overall performance and effectiveness of the system.

4.2 Security Domains and their Administration

The authority delegates the implementation of security policy to a system administrator. In a large network, there may be a number of administrators responsible for rigid observation of the security policy. The purview of a security administrator is known as a security domain. A security domain is defined as a bounded group of security objects and security subjects to which applies a single security policy implemented by a security administrator.

The security domain is a managerial/control concept that defines the scope of a particular security policy [3]. Where the number of security subjects and objects is large, they may be formed into subgroups for ease of management. Such a subgroup is referred to as a subdomain. Normally, the policy of the overall domain will apply to all subdomains. Thus a domain covers all or part of a given distributed system.

One authority will dictate policy for one domain, and another authority will dictate policy for another domain. A successful association should only be possible if the security policies, services and mechanisms of both end systems are compatible. Although there is no logical difference between local activities and remote activities, a local activity may be assured of compatibility within a security policy local to the domain, whereas a remote activity may require interdomain "translation" protocols to ensure effectiveness of an overall security policy. This may lead to incompatibility between domains. In this event, the incompatibility is arbitrated and resolved by reference to a higher authority. These higher authorities may take the form of regional and then national committees, that must meet given codes of practice, contractual specifications, or the ISO standards. Any authority dictating policy, not conforming to these standards will by default exclude itself from connectivity within the complete open security framework.

Within each domain, the security administrator is responsible for the implementation of the domain policy and for assuring its continued effectiveness. This responsibility includes the installation of trusted hardware and software functionality, monitoring day-to-day operations, and recovery in case of breach of security or fault conditions.

A logical model can be constructed with a defined hierarchy where each entity within the model will

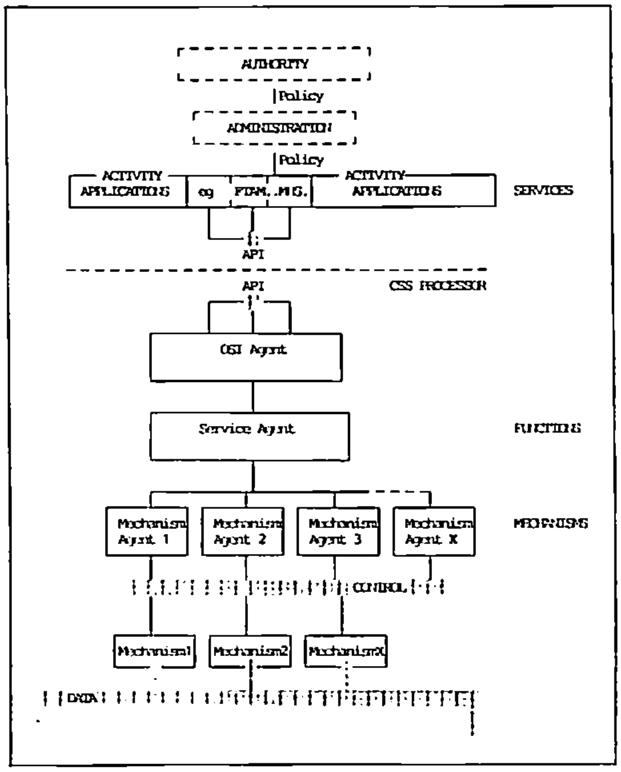


Fig. 1. Security object hierarchy within security domain (information data).

have specific tasks to perform under the purview of its superior, see Fig. 1.

Within this model, any user entity or application entity that is allowed by the security policy to access the security services can obtain/provide information securely to other authorized users within the ODP environment. A user entity may request the access of an object or service in normal (insecure) mode (either accidentally or intentionally), but if this object or entity is itself subject to the security policy, then that policy will force the security services to be invoked for this activity, or access will not be possible at all. This approach to security policy will account for both human error, and attempts at criminal misuse of the system.

5. Conceptual Model of CSS Processor

Essentially, the CSS coexists with the application entities it is to protect. Within the context of the OSI model, the CSS will reside within the application layer as another entity (see Fig. 2). The CSS has access to both the calling application and the application user to request information when required, and these in turn have access to the CSS to invoke functions when required.

The CSS offers the application or user entity a number of security services, which the entity must access through a standard application program interface (API) to the CSS. The API is provided by the CSS and generally depends on the system environment upon which the CSS is hosted. Application entities must provide the correct format data to meet the requirements of this interface in order to take advantage of the CSS as a value-added service. The interface consists of a set of service and associated parameters used by the application and CSS.

In practice, the CSS may comprise a trusted, tamper-proof hardware module, and associated software. The security processor for a large distributed system, need not physically reside in one location, but may be itself distributed throughout the system. Indeed, such an arrangement may be advantageous. One approach is to adopt the concept of a security management centre (SMC), which acts as a central security "exchange", and which will be responsible for the management and control of secure activities on the network. This will include duties such as third-party provision and verification of public keys; notarization, registration and certification services, and association policing to ensure the integrity of a secure communication between two users throughout the duration of that association.

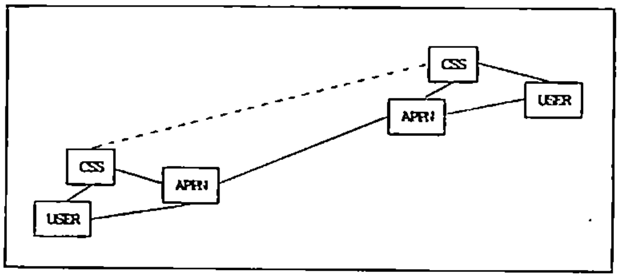
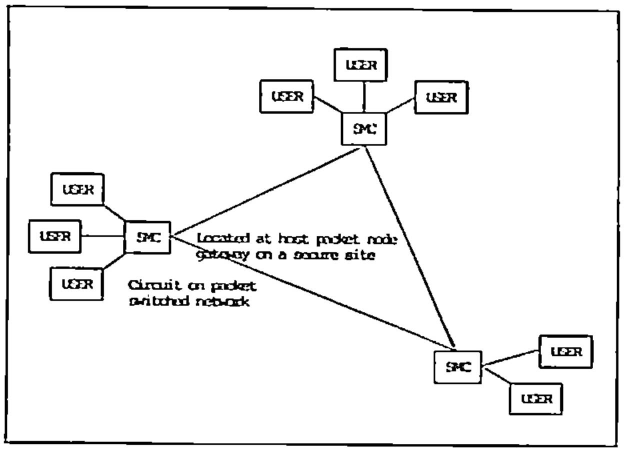


Fig. 2. Relation of the CSS to users and applications.

Security in very large systems may be implemented and controlled with the aid of distributed security processors based in security management centres (SMC), analogous to packet-switching centres in data networks [4]. Each SMC would fully control a number of user terminals hosted upon it, determining authentication, general user access rights and privileges which the SMC would hold in its security management information base (SMIB). Secure communication across the network would involve protocols linking each end user terminal to the host SMC, and protocols linking SMC to SMC (see Fig. 3). The system would provide full flexibility of services irrespective of the location of the user within the network.

5.1 Security Services Supported by the CSS

Although the CSS is implemented as an application entity, it offers a full OSI-wide flexibility owing to the interface architecture. It is very important both from the conceptual and practical points of view, to appreciate the "vertical" structure of the pro-


USER
USER
USER
SMC
USER
USER
SMC
Located at host packet node gateway on a secure site
USER
Circuit on packet switched network
SMC
USER
USER
Fig. 3. The use of security management centres.

posed interface. An advantage of this system is the potential flexibility due the possibility of the CSS functions being called by other than operations in the application layer. The CSS API, which could take the form of a software interrupt, for example, is accessible from any of the OSI layers, not just the application layer. It is quite permissible for the transport layer, for example, to request data encryption services from the CSS. This conforms with the recommendations of OSI, which states that while the majority of security functions can be carried out at the application layer, there are a few which may need to be implemented in different layers.

The CSS supports, among others, the following security functions:

(1) Invocation
(2) Identification
(3) Authentication
(4) Key generation
(5) Key distribution
(6) Encryption
(7) Decryption
(8) Signature
(9) Verification

but the full range of OSI recommendations in security [2] should be possible.

Many different security needs can be met by the concept of a common set of security agents provided externally to the application processes. An agent is defined to be a logical component of the security system, designed to implement a particular function or group of functions. The functional modularization of the system in this manner makes possible the general definition of a flexible security architecture. These agents will be involved with the interactions between users and applications, and the interaction of applications. These agents, their interaction and management are central to the concept of the CSS to be described. A security model comprising ten such agents is suggested.

The generic structure of the CSS, showing the vertical nature of the API, and hierarchy of the internal agents, is shown in Fig. 4.

6. The Agents of the CSS (see Fig. 5)

6.1 User Agent (UA)

The user agent (UA) of the CSS comprises "half" the interface to the CSS as seen by the user entity. The CSS can select which UA will be utilized, either the application UA or the CSS UA itself, dependent on the state and requirements of both the user and the CSS. This will occur when the application has requested a security service from the CSS, and the CSS requires some information directly from the user, such as a password, etc.

The main functions of the user agent (CSS) can be considered:

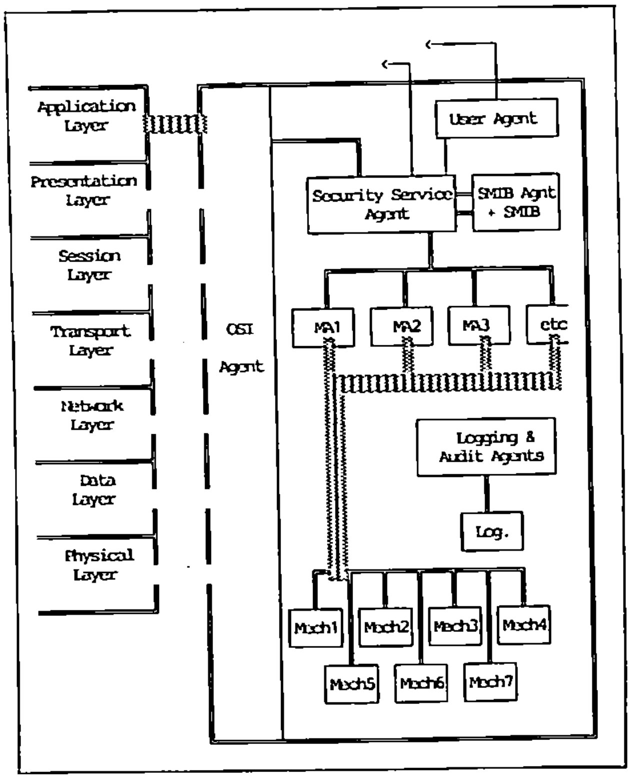


Fig. 4. Generic model of the CSS.

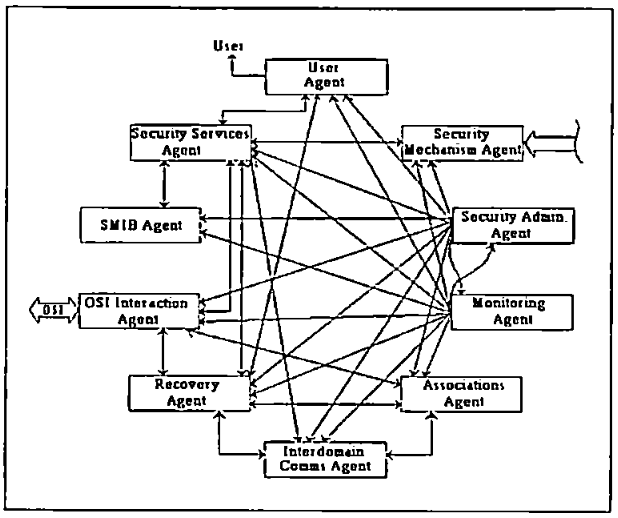


Fig. 5. Interactions between the 10 agents of the CSS.

(1) To interface between the user entity and the security service agent (SSA) of the CSS;

(2) To maintain a library of user entity request statements, via which the UA will determine request/response validity, and suitable responses to the user entity according to a strict set of rules, thus limiting the number of possible user actions;

(3) To interpret all data from the user entity and ensure its validity before presenting it to the SSA, and to determine the location and nature of errors, and inform the SSA accordingly. Also, to process all data from the SSA into a form suitable for interpretation by the user entity before presentation to the user entity;

(4) To accept a request from the SSA when the service requested required further information from the user, and to act upon this to interrogate the user in a suitable manner for this information;

The UA is conceived as a separate entity from the SSA because the UA must be capable of interfacing with many user entities, thus freeing the SSA from the complexities of multiple interfaces.

6.2 Security Services Agent (SSA)

The CSS security service agent (SSA) is the central control agent of the CSS. It is responsible for:

(1) Accepting and checking the validity of all CSS service requests from the OSI agent (OSIA). This is an important function which is necessary to prevent invalid calls from trying to confuse or subvert the SSA into performing functions which are not permitted by the security policy. This validation is accomplished by the SSA checking all requests for security services against the user entity capabilities and privileges stored in the SMIB and the sequence of operations carried out to that time. Any request not expressely permitted for that user entity by the SMIB will be refused.

(2) Selecting the appropriate service mechanisms pertinent to the function to be performed under the supervision of the SMIB via the SMIB Agent (SMIBA), passing the relevant subfunction control data to the service mechanism agents, and sequencing the service mechanism agents correctly to perform the requested security service.

(3) Ensuring the correct routing of the information data to and from, and in the correct sequence among, the security mechanisms. It is possible to implement two completely different security services with the same set of security mechanisms, but merely used in a different order. For example, two security mechanisms implementing a compression (hash) function such as DES in block chaining mode, and an RSA encryption/decryption scheme could be used for (1) a hybrid file encryption system for a confidentiality service, (2) file and message signatures for non-repudiation and integrity checks and (3) checking signatures for authenticity and integrity.

(4) Checking with the SMIB the capability of the user entity, and determining whether the user entity has the privilege to execute the requested service.

(5) Switching between application UA and CSS UA.

6.3 *Service Mechanism Agents (SMA)*

The service mechanism agents must:

(1) Accept control commands from the SSA, and select and control the security mechanisms to perform the service requested, including subfunction of multifunction mechanisms. For example, a DES card could perform normal block encryption, block chaining mode encryption and so on.

(2) Return to the service agent status information including details of function performed, status of operation (success, failure) etc.

6.4 SMIB Agent and SMIB (SMIBA)

The SMIB is the "heart" of the CSS and is the most important unit from the security point of view, and must be protected to the highest level of security.

The SMIB will comprise:

The repository in which the CSS maintains all data pertinent to the security function, including such data as identifications of authorized users, authentication data, user entity capabilities and privileges, etc.

The SMIB Agent will be responsible for interfacing the SMIB to the other CSS agents, including:

Accepting and processing all requests from the service agent for information from the SMIB, including such data as user entity identity checks, user entity authorization, user entity capabilities and privileges, object entity validity, object entity authorization, object entity security status etc.

The internal data contents of the SMIB are summarized in Table 1.

System User Entity Data

The SMIB shall hold information on the users' rights to access the system, and their capabilities and privileges within it. The data will include

(1) User system-wide name (unique identifier).

TABLE 1
SMIB internal logical data structure

Information on system users			
Name 1	# Password 1	Extra data	Capabilities
Name 2	# Password 2	Extra data	Capabilities
Name 3	# Password 3	Extra data	Capabilities
Name N	# Password N	Extra data	Capabilities

Sequencing data for security functions	
Function # 1	Sequencing data
Function # 2	Sequencing data
Function # 3	Sequencing data
Function # 4	Sequencing data

Secure store		
Temporary store for sensitive data not yet processed by CSS	Obj # 1	Token
	Obj # 2	Token
	Obj # 3	Token

(2) An encrypted version of the user entities password, for use during authentication. The user entity therefore enters his password, which will be encrypted via a one-way function before comparison. This renders compromise of the SMIB username/password file less useful to an attacker, as it is computationally infeasible to reconstruct the password from the encrypted version. The file itself, however, must be protected to prevent guessing attacks on the passwords, the planting of known encrypted passwords within the file, and possible software attacks such as "worm" programs [5].

(3) Further data known only to the user that may be used for further authentication in the case of highly privileged operations or possible uncertainty of identity (this may be used in future when the system is semi-intelligent and takes account of users' habits. If a user habitually uses the system only in the mornings, then suddenly uses the system one night, the CSS may require further proof of identity than just the normal password). Measures such as these will be in conjunction with the more familiar "unattended session" defences such as timeout after absence of activity.

(4) A capability token summarizing the users' rights and privileges to perform certain operations. This token is passed to the SSA when it needs to validate a user request.

Security Function Sequencing Data
As the CSS can perform a number of functions with a limited number of security mechanisms, then in order to ensure that security services are correctly performed in accordance with security policy, the SSA will control the SMA strictly in accordance with sequencing data held in the SMIB. This will prevent irregular requests (which may have been specifically constructed ambiguously as an attack on the system) from being accepted by the SSA.

Temporary Data Store
For a full-duplex connection, or with a multi-user arrangement associated with the encryption functions, chained or feedback vectors will be required for the two different transmission directions, and possibly for a number of simultaneous connection conditions. This information must be stored until the relevant data arrives. Also, it is possible that the CSS will be interrupted from processing I/O to service a request from another local user entity. Should the CSS be about to encrypt a sensitive piece of data which is still in plaintext, it will store this piece of data within the SMIB to ensure that it is safe until the CSS has time to process it. Similarly, incoming data may also be stored here until they are processed.

System Objects Data
The SMIB is likely to hold data on the general security status of objects within the system (files, databases etc.) in the form of tokens. When a user subject entity wishes to access a system object entity, the user entity is authenticated against his SMIB data as described. The capability of the user entity is also matched against the classification of the object entity to ensure that the user entity privilege is equal or higher.

6.5 Security Administrator Agent (SAA)
The security administrator agent (SAA) is responsible for allowing only the administrator to provide modifications to the existing system. There must be very strict protocols and authentication for this type of operation within the system. The SAA is also responsible for the strict imposition of system security policy upon the individual operation of and interaction between, the other agents of the CSS. The main function of the SAA involves controlling the SMIBA to place information into the SMIB, or modify existing information as new users are added to the system, existing users removed, security policy updated, user capabilities and privileges modified, and adding/modifying mechanisms and services.

6.6 AGENT for Interactions with other OSI Management Functions (OSIA)
The OSI Agent is responsible for:

Accepting security service requests across the API and interpreting, validating and routing the requests to the SAA. Application software packages without the necessary API will not be able to call the CSS in the first place. Those packages with the API which make CSS request calls in error will be returned an appropriate error code by the OSIA. (See example under SSA for the likely types of information and control data to be distributed.)

Ensuring that all output from the CSS, including control and information data, is routed back across the API into the same layer which originated the request and the control/information data, to prevent "short-circuiting" of layers.

It should be noted that this agent is specific to an OSI system implementation of the CSS. In general, the agent will provide the interface to whatever underlying network architecture is in use, and may be designated the external environment agent (EEA).

7. Logging Agent (LA)

The logging agent (LA) is responsible for:

Accepting and processing all data gathered by the service agent and passing it to the SMIB for logging, including such data as security service requested, date and time, calling user id, calling process, status (success, failure) etc. It is envisaged that the LA will itself internally request one of the CSS encryption services, to encrypt the log ready for storage. The only entity with access to the log will be designated levels of system administration, who will possess the decrypting keys for the log. Allowing managerial access to the log for the preparation of audit reports for security management and resource optimization purposes. The system administration will possess access rights to the log data, and such access rights will be stored in the SMIB.

The LA could be an AI-based module that will detect problems and even likely problems before they occur, and take the necessary action for preventative or remedial measures.

8. Recovery Agent (RA)

The recovery agent (RA) is responsible for all system fault protection and CSS error recovery. Faults and errors may be caused either by hardware failure of units both within the CSS and external to it, and also by certain combinations of situations with which the CSS cannot cope, due to ambiguity of requests for example. The RA will perform the important task of detecting these errors, and placing the CSS into such a state as to maintain the security integrity of the system, so that the CSS is not left in a state where it is vulnerable to attack.

Faults outside the CSS could in certain circumstances also produce system errors. For example, an incorrectly constructed or incomplete data structure could be ambiguous, and the CSS may "hang". Internal error recovery routines will automatically re-request the data, but in the absence of response, the CSS will place itself into a stable, secure state. The CSS has in effect, recovered "internally" from the error, but cannot, of course, influence events outside the CSS. "Inside" and "outside" the CSS refer to software modules within the CSS kernel and those outside the kernel respectively. Obviously, in a distributed security system, the boundaries cannot be clearly defined. The external error must be recovered by the run-time library of the application package.

In the case of a fault developing with either the SMIB or the SMIBA, the CSS protocols must be designed such that the CSS recovery agent (RA) always returns fatal errors to the OSIA for ALL requests. Thus failure of the SMIB terminates all security activity on the local terminal. This differs from faults which may develop with other CSS components, which will not return severe errors, but merely limit the operational effectiveness of the system to those functions not requiring the damaged facility.

9. Associations Agent (AA)

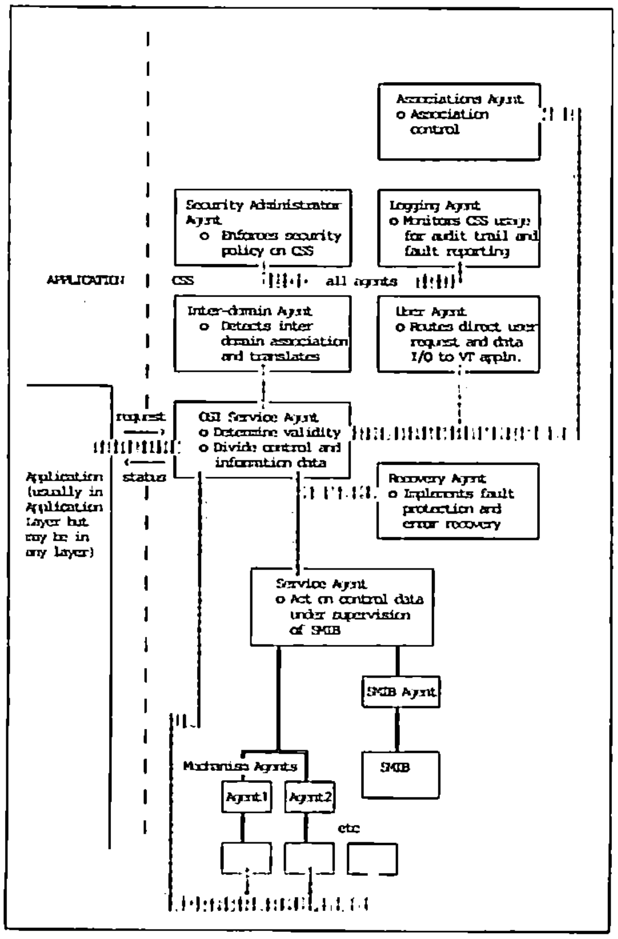
The association agent (AA) is responsible for the security control of the association between remote end user entities throughout the duration of the connection. It is responsible for sending the appropriate data when the connection is set up, such as keys, vectors, time stamps and so on, for exercising supervisory control during the connection, and for clearing down the association from the security facility aspects. In addition, detection of denial of service attacks would be possible with this agent by the sending and receipt of random supervisory packets, subject to the current quality of service conditions.

10. Inter-domain Communication Agent (IDCA)

The connectivity of entities within a domain is assured, as all communicating entities are subject to a common security policy. Inter-domain communication, however, presents a special problem. The communicating entities in the two domains require the use of "translation" protocols to ensure a seamless continuity of security around the association. The inter-domain communication agent (IDCA) is responsible for recognizing inter-domain associations, and invoking additional protocols as necessary. Inter-domain translation may or may not be possible:

For example, the remote entity may be using a form of encryption unknown to the local entities CSS processor. In this case, translation is impossible. The IDCA will note that decryption by the local security mechanisms is not possible, and will flag the appropriate error. The RA will return the appropriate flags to the calling application, which will then either terminate the association, or re-request the remote entity to communicate in another secure mode.

If the remote entity, however, is using say an M-bit algorithm, whereas the local processor is capable of decrypting only N-bit code, then communication is possible, providing the data is reblocked according to the appropriate run length before encoding/decoding. The IDCA is responsible for requesting the appropriate mechanisms via the SSA to attempt the translation. The formal analysis of this kind of system is difficult, but is representative of the sort of real problems that will be encountered when connecting diverse real networks.



APPLICATION
CSS
Associations Agent
o Association control
Security Administrator Agent
o Enforces security policy on CSS
Logging Agent
o Monitors CSS usage for audit trail and fault reporting
all agents
Inter-domain Agent
o Detects inter domain association and translates
User Agent
o Routes direct user request and data I/O to VT appln.
request
status
OSI Service Agent
o Determine validity
o Divide control and information data
Recovery Agent
o Implements fault protection and error recovery
Application (usually in Application Layer but may be in any layer)
Service Agent
o Act on control data under supervision of SMIB
SMIB Agent
SMIB
Mechanism Agents
Agent1
Agent2
etc

Fig. 6. Agent communication within the CSS.

An Example

An example of a very simple "flow" between the application and the agents and mechanisms of the CSS will help to clarify the concepts.

Consider Fig. 6. A typical sequence of component interactions may be as follows:

The application (*e.g.* a secure FTAM or MHS module, see Fig. 7) calls a security function from the CSS (*e.g.* user authentication, encryption etc.) via the API and OSIA [#1]. The application will have set up the control data for the CSS request, along with the information data upon which the CSS will act. The request for the service will be passed over the API (by means of a system interrupt, for example), to the OSIA. The OSIA will note the location of the

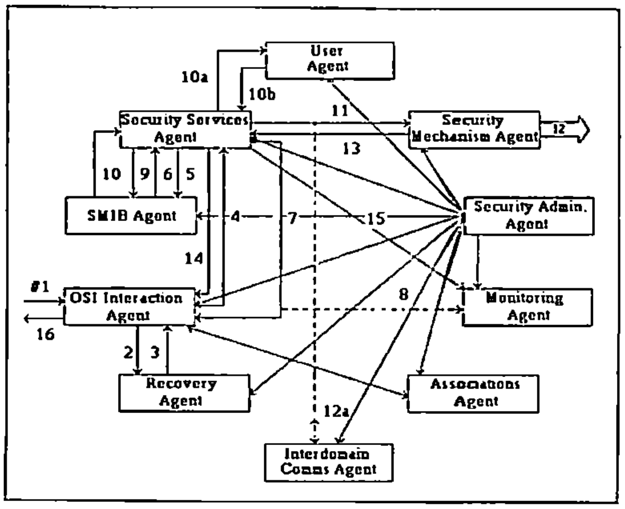


Fig. 7. Interactions between the CSS agents in the secure FTAM example.

origin of the request (*i.e.* the layer where it originated), and attempt to validate the request. Validation is performed at this stage only from the point of view of checking that the request is recognized by the CSS (via a look-up table) NOT to check that the request is permitted. (This is performed later by the SSA and SMIBA). If the data structure is not correctly built, the RA will detect the error [# 2], and initiate a request for a rebuild of the data, and pass this back to the OSIA [#3] for communication with the calling application. If the data are acceptable, the OSIA retrieves the information data and places it onto the internal CSS data bus. The control information part of the request is routed to the SSA. [# 4] The SSA examines the control data, and interrogates the SMIB via the SMIBA [# 5] [# 6] as to whether the request is (a) valid, (b) legal.

If the request is INVALID for either reason, the SSA will return the appropriate error code to the OSIA [# 7], which will inform the calling application process accordingly. The LA will note the illegal request [# 8]. If the request is VALID, then the SSA will interrogate the SMIB via the SMIBA [# 9] [# 10] as to the correct sequence of operation to be performed to execute the requested function in line with security policy. If the SMIBA has informed the SSA that, for example, an additional password is required from the user, then the SSA will request this from the UA [# 10a, # 10b]. The SSA will select and sequence the appropriate SMA's [# 11] to fulfil the function, passing each control data as necessary, which in turn will select and sequence the appropriate security mechanisms [# 12] which will act upon the information data on the bus which is to be processed. (The IDCA will have also noted that this is a local operation, and no inter-domain activity is needed [# 12a]). Upon completion, (signalled by each mechanism agent in turn to the SSA [# 13]), the SSA will indicate task status to the OSIA [# 14], and also inform the LA of the function performed [# 15], for whom, and the task status. The OSIA will then retrieve the processed information data from the CSS bus, and return the processed data across the API to the correct calling location, along with the function status [# 16].

Within the framework of the ten-agent model, an implementation for the IBM PC has been written using Pascal and 8086 assembler. A file encryption and transfer (FET) has been coded, and tested between machines via serial port communications. It is planned to extend this to an Ethernet LAN in the near future, with several users making secure transfers simultaneously.

11. Conclusion

There is a need to extend OSI to meet the recommended security services and others being proposed. The described system will implement the security function efficiently but requires some modifications to existing applications. The API can be defined for the general case, but interface details are highly implementation specific.

The concept of an existing network with security facilities managed by a number of security management centres seems appropriate for large networks and mobile users. The idea of a number of agents to fulfil the security functions has been proposed for flexibility and ease of implementation.

Acknowledgments

The authors wish to thank the IBM (U.K.) Laboratories for their sponsorship, and colleagues on the CEC Cost 11-ter, Security in Computer Networks Committee for their collaboration.

References

[1] ISO/IEC JTC1, *ISO Technical Report on Security in Open Systems*, ISO, June 1988.

[2] ISO 7498-2, *Open Systems Interconnection: Basic Reference Model—Security Architecture*, ISO, February 1989.

[3] ISO/IEC JTC1/SC21, *Working Draft: Access Control Framework*, ISO, December 1988.

[4] S. Muftic, (ed.), *Security Mechanisms for Computer Networks*, Ellis Horwood, Chichester, 1989, ISBN 0-7458-0613-9.

[5] E. H. Spafford, The Internet worm program: an analysis, *CDS-TR-823*, Purdue University, IA, November 1988.

Simon J. Shepherd held a Commission for 12 years in the Royal Navy, and gained a First Class Honours Degree in electrical engineering at the Royal Naval College Manadon in 1986. On leaving the Royal Navy, he held the position of Senior Systems Engineer in the Special Studies Group at British Aerospace plc, where he was responsible for modelling and simulation of guided weapons systems. He is currently with the Network Research Group at Polytechnic South West, where he is investigating security in large open systems for the IBM United Kingdom Laboratories, and is registered for a Ph.D. with the CNAA.

Peter Sanders is the Assistant Head of Department of Communication Engineering at Polytechnic South West, Plymouth, and the leader of its Network Research Group. He is currently involved with research into security systems within networks and techniques for the design and optimization of switching systems, in conjunction with major U.K. companies. He is a member of the CEC COST II "Security Mechanisms for Computer Networks" project.

Ahmed Patel received his M.Sc. and Ph.D. degrees in computer science from Trinity College, University of Dublin, in 1977 and 1984 respectively. He specialized in the analysis, design and implementation of communications protocols for computer networking. From 1979 to 1982 he was responsible for developing the Experimental Irish Universities Data Network. From 1984 to 1986 he was responsible for developing the EuroKom computer conferencing and electronic mail service which is used by ESPRIT participants. He currently lectures in the Department of Computer Science at University College Dublin. His special fields of research interest include computer networks, conformance testing of communications protocols, network security, network management, broadband communication networks, message handling and directory systems, distributed processing and real-time multitask operating systems. Dr. Patel has published many technical papers in these areas and he is active in a number of technical working groups in the COST-11, RARE, RACE and ESPRIT communities.

The Quadratic Residue Cipher and some Notes on Implementation

S J Shepherd*, P W Sanders** & C T Stockel**

* Electrical Engineering Department
University of Bradford
Bradford, UK

** Network Research Group
Faculty of Technology
University of Plymouth
Plymouth, UK

ABSTRACT. Although of similar age, the Quadratic Residue Cipher (QRC) has been neglected compared with the publicity received by other public key cryptosystems, notably the RSA cipher. This paper attempts to redress the balance somewhat, explaining in expository form the principle of the QRC, the advantages it offers over RSA and some experiences gained as a result of using the cipher.

INTRODUCTION

Since Diffie and Hellman published their famous paper [1] in 1976 which introduced the concept of a 'public key' cipher, cryptography has undergone a revolution. The possible elimination of the need to distribute pairs of secret keys between communicants, with the associated logistical and security problems, has led to wide interest in the use of public key systems. The benefits are especially apparent in large systems, where the quadratic explosion in the number of symmetric key pairs needed for a classical (symmetric) cipher system can be a security management nightmare.

One of the first practical public key implementations was invented by Rivest, Shamir and Adleman [2] and named after them (RSA). Their system has become by far the best known and most publicised of the public key systems. It is by no means the only one, however, a large number of others having been described in the literature, such as those by Rabin [3] and Merkle and Hellman [4]. These systems vary from elegant but useless mathematical curiosities, to those which are now in current use. One of the most potentially useful systems however, is the Quadratic Residue Cipher (QRC), which has received surprisingly little attention since it was first described by Blum, Blum & Shub [5]. This may be due in part to the limitation that it is a secrecy only system, whereas RSA can be used for both secrecy and authentication (digital signatures). In the context of a pure secrecy system though, it offers a number of significant advantages over RSA in both implementation and ease of use [6]. Of greater potential significance, is the recent proof [7] that the difficulty of breaking the QRC is equivalent to the difficulty of factoring. This is not the case for RSA, and some reports [6] even go as far as to suggest that RSA may have hidden weaknesses as yet undetermined.

PRINCIPLE OF QRC

The QRC relies upon the principle that it is easy to determine the square of an integer modulo n, but finding the square root of a (large) integer modulo n is intractable. Also, not all such integers have square roots. Those numbers having roots are known as *quadratic residues* and those which do not are known as *quadratic non-residues*. More formally, if n is a positive integer, then a is a quadratic residue of n if $(a,n) = 1$ and if the congruence $x^2 \equiv a \pmod{n}$ has a solution. If the congruence $x^2 \equiv a \pmod{n}$ has no solution then a is a quadratic non-residue of n.

For example, to determine the quadratic residues of 11, we compute the squares of the non-zero elements in the field modulo 11, i.e. the integers 1 to 10. This gives

$$\begin{aligned}
1^2 &\equiv 10^2 \equiv 1 \pmod{11}\\
2^2 &\equiv 9^2 \equiv 4 \pmod{11}\\
3^2 &\equiv 8^2 \equiv 9 \pmod{11}\\
4^2 &\equiv 7^2 \equiv 5 \pmod{11}\\
5^2 &\equiv 6^2 \equiv 3 \pmod{11}
\end{aligned}$$

Although we have squared all ten non-zero elements in the field (1..10), only 5 squares result (1,3,4,5,9). The quadratic residues of 11 are therefore 1,3,4,5,9 and the quadratic non-residues of 11 are those numbers which are left (2,6,7,8,10). The fact that there are the same number of residues as non-residues is not a coincidence. It can be shown that if p is an odd prime, then there are exactly as many quadratic residues as non-residues of p among the integers $1, 2, \ldots, p\text{-}1$. Proofs of this can be found in most books on number theory. The important point to note, however, is that in the field (designated $\mathbf{Z}_p$ for prime p) where $p \equiv 3 \pmod{4}$ such as $p = 11$ in this case, only the quadratic residues have square roots one of which is itself a quadratic residue. This root is known as the *principal root* of the residue. In the above example, 4 has square roots 9 and 2 but only 9 is a residue with two square roots of its own. Of these roots, 3 and 8, only 3 is a residue and so forth. (In the case where $p \equiv 1 \pmod{4}$ this is not always true. In $\mathbf{Z}_{17}$, the residue 16 has square roots 4 and 13 *both* of which are residues with square roots.)

Encryption

To construct the QRC, we proceed as follows:

The recipient **R** chooses two large prime numbers p and q using one of the many tests available [e.g. 8], subject to them both being congruent to 3 mod 4. This is in no way restrictive as it can be shown that half of all the primes of a given length will satisfy this condition [9]. These two primes become **R**'s secret key. **R** then multiplies p and q to produce his public key n,

$$\text{Public Key} \quad n = pq$$

Such an integer which is the product of two primes both congruent to 3 mod 4 is known as a *Blum* integer. The public key (which the reader will observe is simpler to produce than the RSA key pair) is made available and anyone wishing to send **R** a message proceeds as follows:

The sender S chooses any quadratic residue in $\mathbf{Z}^*_n$. (For composite *n*, the set of elements *i* where $i < n$ is designated $\mathbf{Z}_n$, and the subset of these which are quadratic residues is designated $\mathbf{Z}^*_n$). Although it is possible to test whether a given number is a quadratic residue (by *quadratic reciprocity*), the easiest way to obtain a quadratic residue is simply to pick any number relatively prime to *n* and square it, modulo *n*. This residue becomes the 'seed' x_0 for the key generator which will be used to encipher the message.

S adopts some pre-arranged scheme to assign binary values to the residues, such as writing a 1 when the parity is even, and a 0 when the parity is odd. To proceed, S writes down the appropriate key bit according to his first residue x_0, and then squares this residue. S then writes down the next corresponding bit, and squares the resulting residue again, and so on. In this manner, a pseudo-random binary bit pattern is produced which can be used as a 'one-time pad' to encrypt the message, providing of course, that the seed is not reused with the same modulus. The significance of a one time pad is that such a cipher is the only one providing complete *theoretical* security.

The bit pattern produced by this type of 'square mod *n*' generator has been shown [5] to exhibit very 'random' properties, and is cryptographically strong. (Tests of randomness usually require such sequences to pass certain statistical tests [8], such as the long-term frequency of 0's and 1's should be similar, and that the 0's and 1's should be 'well mixed', However, statistical tests are not sufficient in all cases. An important property of cryptographically strong random sequences is that they should be unpredictable. A pseudo-random sequence generator is generally defined [10] to be cryptographically strong if the sequences it generates are not predictable in the sense that given an arbitrarily long sample of the sequence, it is not possible to obtain any more knowledge about the missing bit either to the right or to the left than could be gained by tossing a fair coin.)

More detailed analyses [11,12] show that one can use more than the parity bit after each squaring operation. There is no apparent weakening of the resulting sequence if the first $\log_2 l$ least significant bits are used, where *l* is the bit length of the residue. For example, if the residues were 8 bit words, then the $\log_2 8 = 3$ least significant bits could be added to the key stream from each residue reducing the number of squaring operations required by a factor of three. Since the key sequence will be used as a 'one-time pad', for security it is important that the sequence have as long a period as possible, certainly longer than the message to be encrypted. The period of the quadratic residue generator, however, is not easily determined. It is *usually* equal to $\lambda(\lambda(n))$, where λ is the Carmichael function [13]. This important parameter is discussed later and the conditions when it holds are defined.

Having obtained the key sequence, the sender then simply exclusively OR's (XOR) the plaintext (in ASCII form, say) with the key 'one-time' pad and appends the final residue x_N.

Decryption

The legitimate recipient **R** can easily recover the plaintext, because knowing the two (secret) factors of the modulus *n*, he can compute the residue x_{N-1} by taking the square root of the residue x_N, then the square root of x_{N-1} and so on right back to the residue x_0. Knowing the residues, **R** has the key sequence and can therefore recover the plaintext simply by XORing the key with the bit stream portion of the ciphertext. Taking square roots, however, is computationally intensive and of the same complexity as the modular exponentiation required by RSA. Using this simplistic approach gives little benefit over RSA in decryption.

Fortunately, knowledge of the factors of n allows random access both forwards and backwards through the sequence. Knowing the final residue x_N and using the following lemma, it is possible to take the 2^N -th root directly and so obtain the initial seed residue x_0 immediately. **R** then has no more work to do than **S** in recreating the key sequence.

Lemma. Let p be an odd prime and $(a,p) = 1$. Then a is a quadratic residue if and only if $a^{(p-1)/2} \equiv 1 \pmod{p}$.

Proposition. If the prime $p \equiv 3 \pmod{4}$ and a is a quadratic residue, the roots are $\pm a^{(p+1)/4}$. The principal root will use the plus sign.

Proof. $(a^{(p+1)/4})^2 \equiv a^{(p+1)/2} \equiv a^{(p-1)/2}a \equiv a \pmod{p}$.

Corollary. By induction, if $p \equiv 3 \pmod{4}$ and a is a 2^N -th power, then the 2^N -th root of a is

$$a^{\left(\frac{p+1}{4}\right)^N}$$

The reason for the original constraint that the factors of n must be congruent to 3 mod 4 is now clear. Only if $p \equiv q \equiv 3 \pmod{4}$ will the power be an integer when it can be immediately evaluated.

An eavesdropper **E**, however, cannot decipher the message because **E** does not know the factors of n and calculating them is intractable. Without the factors, **E** cannot take square roots in $\mathbf{Z}^*_n$, and hence cannot recreate the key sequence.

An example

For the sake of clarity in our example we will use very small numbers and a very short message. In practice, the numbers involved will be larger, on the order of 100-200 digits to discourage factoring, and the messages will normally be correspondingly longer.

Let **R** choose two primes p and q, remembering that they must both be congruent to 3 modulo 4, say,

$$p = 11$$
$$q = 23$$

R multiplies these to give his public key n

$$n = 11 * 23 = 253$$

which is made available.

S wishes to send **R** a message 16 bits long, say the letters 'QR'. The 8-bit ASCII representation of this is

Plaintext = 0101000101010010

S picks a random number (relatively prime to 253), say 20, and squares this modulo 253. This yields the first quadratic residue $x_0 = 147$. This residue is successively squared until sufficient residues have been created to encrypt the plaintext:

$$
\begin{aligned}
x_0 &= 147\\
x_1 &= 104\\
x_2 &= 190\\
x_3 &= 174\\
x_4 &= 169\\
x_5 &= 225\\
x_6 &= 25\\
x_7 &= 119\\
x_8 &= 246\\
x_9 &= 49\\
x_{10} &= 124\\
x_{11} &= 196\\
x_{12} &= 213\\
x_{13} &= 82\\
x_{14} &= 146\\
x_{15} &= 64
\end{aligned}
$$

S then writes down the least significant bit(s) of each residue. The key sequence is therefore

Key = 1000111101001000

The key pad is then XOR'ed with the plaintext to produce the ciphertext:

Key	= 1000111101001000
Plaintext	= 0101000101010010
Ciphertext	= 1101111000011010

Finally, the last residue (64 decimal = 1000000 binary) is appended to the ciphertext to produce the finished message. Alternatively, the residue may be sent via other channels.

Ciphered message = 1101111000011010 [*delimiter*] 1000000

The message can now be sent over public (insecure) channels to the recipient **R**, who proceeds to decipher the message as follows.

First, **R** notes that the length of the message is N+1 = 16 bits. **R** therefore knows that the original seed residue x_0 must have been squared N = 15 times to produce the final residue x_N. Now **R** could take successive square roots 15 times to produce each intermediate residue but, as we have stated, this is computationally excessive and access to any ***random*** point in the sequence is possible given the factors of *n*. **R** can therefore take the 2^{15} -th root directly and obtain the seed residue x_0 immediately.

For completeness, we demonstrate both the simple approach using repeated square roots, and then demonstrate the more efficient 'direct root' method. Finally, we will describe a cleverer algorithm which computes the 2^N -th root directly, while considerably simplifying the computations.

Algorithms for decryption

Taking the square root of the final residue x_{15}, that is, $\sqrt{64} \pmod{253}$, we compute using the prime factors of *n*

$$64^{(11+1)/4} \pmod{11} = 3$$

$$64^{(23+1)/4} \pmod{23} = 8$$

The required root x_{14} is therefore congruent to 3 (mod 11) and also congruent to 8 (mod 23). We now use the *Chinese Remainder Theorem* (found in most texts on elementary number theory) to compute the value of the root which satisfies both these congruences (mod 253). The answer is 146, which is the required root x_{14} (compare with the table on page 5). We could now repeat this process a further 14 times to recreate the key sequence but as mentioned earlier, it is easier to compute the $(\frac{1}{2})^{15}$ -th power of the residue directly.

We have from the previous corollary

$$64^{\left(\frac{11+1}{4}\right)^{15}} \pmod{11} = 4$$

$$64^{\left(\frac{23+1}{4}\right)^{15}} \pmod{23} = 9$$

Applying the Chinese Remainder Theorem to these results, we find that the residue x_0 is 147 which is the required result, compare again with the table on page 5.

The clear disadvantage of this method is the problem of computing to the enormous powers involved. Even the use of efficient algorithms such as the *repeated squares* method [8] still results in inefficient computation.

A great improvement lies in the observation that

$$a^b \pmod{m} = a^{b \pmod{\phi(m)}} \pmod{m}$$

This considerably reduces the amount of effort required provided that $\phi(m)$ can be found. In our case though, m is prime whence $\phi(m)$ follows immediately as it is simply equal to m-1. Using this principle, we can construct a very efficient algorithm [6] as follows.

When setting up the cryptosystem the recipient **R**, knowing the secret factors p and q, uses Euclid's Extended Algorithm to precalculate two quantities a and b, such that

$$ap + bq \equiv 1 \pmod{n}$$

On receipt of a message, knowing its length N+1 and the final residue x_N, **R** proceeds to compute

$$\sigma = [(p+1)/4]^N \pmod{[p-1]}$$

$$\theta = [(q+1)/4]^N \pmod{[q-1]}$$

$$u = (x_N \bmod p)^\sigma \pmod{p}$$

$$v = (x_N \bmod q)^\theta \pmod{q}$$

$$x_0 = (bqu + apv) \pmod{n}$$

Using this algorithm in our example, **R** would have found

$$11a + 23b \equiv 1 \pmod{253}$$

whence $a = 21$ and $b = 1$.

R knows the residue x_N corresponds to the 15-th squaring operation, so he computes

$$\sigma = [(11+1)/4]^{15} \pmod{10} = 3^{15} \pmod{10} = 7$$

$$\theta = [(23+1)/4]^{15} \pmod{22} = 6^{15} \pmod{22} = 10$$

$$u = (64 \bmod 11)^7 \pmod{11} = 9^7 \pmod{11} = 4$$

$$v = (64 \bmod 23)^{10} \pmod{23} = 18^{10} \pmod{23} = 9$$

$$x_0 = ((1 * 23 * 4) + (21 * 11 * 9)) \pmod{253} = 147$$

which is the required result!

The algorithm has simplified and speeded up the calculation of, for example

$$64^{(5^{15})} \pmod{23}$$

by replacing the computation with much more manageable quantities such as 18^{10} (mod 23). The efficiency is compounded as the numbers get larger and is indispensable with the realistically secure numbers used in practical systems.

Calculation of the Key Periods of the x^2 mod N Generator

The critical parameter of the QRC key generator is the *period* of the key sequence. In this expository paper we quote the important results without proof. For a complete treatment of the background, derivations and proofs of these results the reader is referred to the seminal paper of Blum *et al* [5].

For the convenience of the reader, we summarise a few basic number theoretic ideas that are needed in the derivation of the period.

Universal exponents

A universal exponent is a non-zero quantity U such that $a^U \equiv 1 \pmod{m}$ for all a with $(a,m) = 1$. For example, the Euler Phi function $\phi(m)$ is a universal exponent. The *least positive* universal exponent is called the Carmichael function of m, written $\lambda(m)$.

A definition [13] of the Carmichael function is as follows

$$\lambda(p^a) = \phi(p^a) \quad \text{if } p=2 \textit{ and } a \leq 2, \textit{ or } p \geq 3$$

$$\lambda(2^a) = \tfrac{1}{2}\phi(2^a) \quad \text{if } a \geq 3$$

$$\lambda(m) = lcm\,[\lambda(p_i^a)] \quad \text{otherwise}$$

where p_i^a is the prime power factorization of m. Knuth proves [8] that $\lambda(m)$ is both the least common multiple and the supremum of the orders of the elements in Z^*_m. As a corollary, we have Carmichael's extension of Euler's theorem

$$a^{\lambda(m)} \equiv 1 \pmod{m} \text{ whenever } (a,m) = 1$$

As stated on page 3, the period of the QRC generator is *usually* equal to $\lambda(\lambda(n))$ where n is the public key modulus. We quote the sufficient conditions for this later, but applying the expression to our earlier example we have $\lambda(253) = lcm\,[\phi(11),\phi(23)] = lcm\,[10,22] = 110$. $\lambda(110) = lcm\,[\phi(2),\phi(5),\phi(11)] = lcm\,[2,4,10] = 20$. Therefore $\lambda(\lambda(253)) = 20$ and if the sequence in the example is carried forward the residues indeed repeat with period 20.

Primitive Roots and Orders

If $(a,n) = 1$ then the least positive x such that $a^x \equiv 1 \pmod{n}$ is called the *order* of a mod n and we write $\text{ord}_n\, a = x$.

For example, to find the order of 2 mod 7, compute the powers of 2 mod 7

$$2^1 \equiv 2 \pmod{7}$$
$$2^2 \equiv 4 \pmod{7}$$

$2^3 \equiv 1 \pmod 7$
$2^4 \equiv 2 \pmod 7$
etc

The smallest power congruent to 1 is 2^3. Thus $\text{ord}_7\, 2 = 3$.

Lengths of Sequences Produced by the x^2 mod N Generator

Let $\pi = \pi(x_0)$ denote the period of the key sequence with seed x_0. In [5] it is shown that if $n = pq$ where p,q are *distinct* odd primes both congruent to 3 (mod 4) then $\pi | \lambda(\lambda(n))$. It is then shown that the sufficient conditions under which the converse is true, that is, when $\lambda(\lambda(n)) | \pi$ and hence when $\pi = \lambda(\lambda(n))$, are when the following are met:

1. **Choice of modulus n** (Theorem 7 of [5])

Choose n such that $\text{ord}_{\lambda(n)/2}\,(2) = \lambda(\lambda(n))$. It is possible to contrive an n to fulfill this requirement by setting $p = 2p_1 + 1$ and $p_1 = 2p_2 + 1$, and similarly for q_1 and q_2 (Theorem 8 of [5]). In our earlier example $11 = 2 * 5 + 1$ and $5 = 2 * 2 + 1$. Similarly $23 = 2.11 + 1$ and $11 = 2.5 + 1$.

Aside from being one of the conditions that ensures the period equals $\lambda(\lambda(n))$, this construction of n also allows us to calculate the period! Given a sufficiently large n without its factorization, evaluation of $\phi(n)$ and hence $\lambda(n)$ is as intractable as factoring n. Knowing p_2 and q_2, however, the period follows immediately. $\lambda(n) = lcm\ [2p_1, 2q_1] = 2p_1q_1$ and so $\lambda(\lambda(n)) = lcm\ [2p_2, 2q_2] = 2p_2q_2$.

2. **Choice of seed x_0** (Theorem 7 of [5])

Choose x_0 such that $\text{ord}_n\, x_0 = \lambda(n)/2$. We can always find a residue to satisfy this condition. Specifically, the number of quadratic residues in Z^*_n that are of order $\lambda(n)/2$ mod N is around $O\,(n\ /\ (\ln \ln n)^2)$ [5].

When the above two conditions do not apply, there seems to be no simple method of evaluating the sequence length π. In these cases, the sequence length is always a divisor of $\lambda(\lambda(n))$ and hence very much shorter than when the above conditions hold. Analysis of the lengths of these sequences is, however, of no practical importance from our point of view since these sequences are to be avoided for cryptographic purposes anyway.

COMPARISON OF RSA AND QRC

Recall that the RSA cryptosystem operates as follows:

An intended recipient **R** chooses two large primes p,q and calculates their product $n = pq$. A preferred constraint on p,q is that they are *strong* [14] in the sense that their product n is more difficult to factor than if p,q were just chosen at random. Basically, this means ensuring that p′ and p″ (where $p' = p - 1$ and $p'' = p' - 1$) both have *at least one large factor*, and similarly for q' and q''. Knowing p,q, **R** can evaluate $\phi(n) = (p-1)(q-1)$. **R** then chooses a quantity e relatively prime to n and calculates a quantity d such that $de \equiv 1 \pmod{\phi(n)}$. **R** than makes the public key $\{e,n\}$ available, and keeps the secret key d private.

A sender S wishes to transmit a message to **R**. S encodes his message by a pre-arranged scheme such as A=1, B=2 etc, and breaks his message up into blocks b_i smaller than the modulus n. S computes the ciphertext c_i from each plaintext block b_i using

$$c_i = b_i^e \pmod{n}$$

On receipt, **R** recovers the plaintext using

$$b_i = c_i^d \pmod{n}$$

We now compare and contrast some aspects of the RSA and QRC systems. Again, we state results here - for detailed arguments and proofs the reader is referred to [6] and the many excellent references contained therein.

Implementation complexity

The QRC key pair is simply $\{n\}$ for the public part and $\{p,q\}$ for the secret part, whereas the RSA key pair as shown above is somewhat more difficult to calculate. This slight disparity is offset by the fact that, in general, the key pair only need be produced once or at intervals. The actual encryption efficiencies in terms of number of arithmetic operations, however, are quite different.

Consider an RSA cryptosystem using a 64 bit modulus and so each message block will be no longer than 64 bits. Suppose that the encrypting exponent e is a 16 bit quantity chosen to be a prime, such as $2^{16} + 1$ which is 10000000000000001 binary. This form of exponent maximises the efficiency of the encryption because where a *zero* occurs, the repeated squares modulo exponentiation algorithm only need perform one modulo multiplication but where a *one* occurs two such operations are required. For an optimum 16 bit exponent therefore, encryption of the block will require 18 modulo multiplications.

In a QRC cryptosystem, the encryption of the 64 bit message block will require the generation of 64 bits of key. If the modulus is again 64 bits long, $\log_2 64 = 6$ bits may be appended to the keystream from each residue. Each residue is generated by a single modulo multiplication. Therefore, only 11 modulo multiplications are required to generate sufficient key length to encrypt the block.

Limitations on plaintext

In RSA the message must be broken up into blocks smaller than the modulus in use. The QRC does not require this to happen, but the overall length of the message must be smaller than the period of the key. In RSA, therefore, messages of unlimited size may be encrypted simply by breaking them into suitable blocks. In the QRC, the message is limited to the period of the key.

Encryption principle

RSA is a *deterministic* cipher in the sense that if an eavesdropper has a candidate for the plaintext (by guessing, say) he can easily verify his guess simply by encrypting it with the public key. The QRC is a *probabilistic* cipher in that an eavesdropper with a candidate for the message *cannot* verify his guess by encipherment because he does not know the starting residue used for the key sequence.

Data Expansion

In RSA, a message block the size of the modulus is encrypted to a block also the size of the modulus. The data therefore undergoes no expansion on encryption. With the QRC, however, the need to append the final residue means that there is a small amount of data expansion on encipherment.

Proof of Security

Even if factoring is genuinely hard, breaking RSA is not known to be equally hard. In other words, it has never been proven that breaking RSA is directly equivalent to the difficulty of factoring. It is possible [6] that d can be computed efficiently from the public key $\{e,n\}$ without the need to factor n. It is also possible that there is another efficient algorithm to recover the message m from e, n and m^e (mod n). Breaking the QRC, on the other hand, has been shown [7] to be directly equivalent to the difficulty of factoring.

Leakage of information

With RSA, even if it turns out that it is impossible to compute all of m from the information available to the eavesdropper, it might still be easy to obtain efficiently some *partial* information such as half the bits in m. This may still be possible even if random padding is used. This phenomenon is known as *partial leakage of information*. The QRC leaks no partial information in this way. The lack of a formal proof of security and the unresolved partial leakage problem are a serious limitation of the RSA system, and may well have contributed to the recent decision by NIST (US National Institute for Standards and Technology) not to adopt RSA.

Resistance to attack

RSA is believed to be resistant to a chosen ciphertext attack, whereas the QRC is *not* resistant to such an attack. A chosen ciphertext attack is one where the cryptanalyst can choose ciphertexts at will, and be supplied with the corresponding plaintexts provided they exist. From this information, he is to infer the decrypting key.

Digital signature

By far the most important advantage RSA has to offer is the possibility of a digital signature. This allows a sender to 'sign' a message using his secret key before encrypting it with the recipients public key. On decryption, the recipient can verify the senders identity by decrypting the signature with the senders public key. The QRC is a secrecy only system, and suffers from the severe drawback of not admitting the possibility of digital signature.

These aspects of the two cryptosystems are summarised in Table 1.

Table 1 - Comparison of aspects of the RSA and QRC Ciphers

Aspect	RSA	QRC
Possibility of Digital Signature	**ADVANTAGE** Digital signature possible.	**DISADVANTAGE** Digital signature not possible.
Existence of a formal proof of security	**DISADVANTAGE** Has *not* been proven to rely on the hardness of factoring, and may have other weaknesses as well.	**ADVANTAGE** Has been proven to rely only upon the hardness of factoring.
Resistance to attack	**ADVANTAGE** Believed to be resistant to a chosen ciphertext attack.	**DISADVANTAGE** Not resistant to a chosen ciphertext attack.
Leakage of information	**DISADVANTAGE** Leakage of partial information even when random padding is used.	**ADVANTAGE** No leakage of partial information.
Algorithm and implementation complexity	**SIMILAR** Requires exponentiation and modulo reduction for both encryption and decryption.	**SIMILAR** Requires similar operations for encryption and decryption but less of them.
Limitations on Plaintext	**ADVANTAGE** Unlimited plaintext size if broken into blocks smaller than the modulus.	**DISADVANTAGE** Plaintext limited by period of key generator.
Encryption Principle	**DISADVANTAGE** Deterministic - only one cipher text corresponds to a given plaintext for a particular key. i.e., an eavesdropper who has a candidate for the plaintext can easily verify a guess simply by encrypting it.	**ADVANTAGE** Probabilistic - many possible ciphertexts can correspond to a given plaintext. i.e., an eavesdropper who has a candidate for the plaintext *cannot* verify a guess simply by encrypting it.
Data Expansion	**ADVANTAGE** No expansion of data when encrypted.	**DISADVANTAGE** Small data expansion when encrypted.

IMPLEMENTATION

A QRC crypto package has been developed in the Network Research Group. The software runs on a standard IBM PC, and offers the following features,

1. **Menu driven** for ease of use;

2. **Key generation** up to 255 decimal digits (840 bits). Given the expected heuristic running time of the best known general purpose factoring algorithm (the *multiple polynomial quadratic sieve*) the expected time to factor integers of this size is roughly equal to the estimated current age of the universe, some 15 billion ($15 * 10^9$) years;

3. **Static encryption and decryption** of data files for storage on disk.

Software language

The main shell of the program is written in Pascal for structural integrity, but all the multi-precision arithmetic routines are written in hand-crafted 80*86 assembly language. The package is therefore quite fast in operation.

Modular structure

The software is written in modular form for flexibility and ease of maintenance. Each module is a stand-alone unit, so that modification of one module does not affect any other. This also allows for improvement in algorithms as they become available and for parts of the system to be written independently by different authors. At present, the system comprises of the following units:

a) Prime generator
b) Key generator
c) Encryption module
d) Decryption module
e) General purpose number theory library

The functions of the first four modules are described in the following discussion. The general purpose library contains all the multi-precision routines called by other modules such as modulo exponentiation, Euclid's algorithm and so on.

Generation of keys

One of the most important modules in the package is the key generator. For security it is very advantageous if the key period is easily determined and, where possible, we have been careful to follow the criteria of [5] discussed above. When the key generator module is requested to produce a key of length l, the steps are as follows:

1. The key generator module requests two primes of lengths $0.45l$ and $0.55l$ from the prime generator module. The reason for these dissimilar sizes as opposed to simply generating two primes of length $l/2$ is to avoid any possibility of the primes being too close together and hence opening up the (remote) possibility of a Fermat attack [15].

2. For each request, the prime generator picks a random odd integer of the required length, and tests it for primality using:

   a) trial division up to 101;
   b) five iterations of Rabin's algorithm [8, *Algorithm P*] to certify probable primality to better than 99.90%.

   If the number fails the tests then it is incremented such that it is neither even nor a multiple of 5 (that is, in decimal notation the number can only end in 1,3,7 or 9). If the probable prime p'' passes the tests, the prime generator then evaluates $p' = 2p'' + 1$ and tests this for primality using the above method. If this also passes, the generator finally evaluates $p = 2p' + 1$ and checks that it is both a probable prime and that it is congruent to 3 mod 4. If this is the case then p is returned along with p'' and q''. If not, then another p'' is chosen and the steps repeated. It is important to note that, apart from returning primes that are suitable for the construction of an analytic QRC generator, this method also has the additional advantage that the primes are *strong* in the sense discussed in connection with RSA [14]. This makes the work of a cryptanalyst in factoring the modulus very much harder than if the primes are chosen at random.

3. The key generator multiples p,q to produce the modulus n.

4. The key generator evaluates $2p''q''$ to produce the key period π.

5. The key generator uses the Extended Euclid Algorithm to compute a,b such that $ap + bq \equiv 1 \pmod{n}$.

6. The key generator stores p,q,a,b,n and π in a disk system file for use in decryption. Note that p,q are not stored in encrypted form. Access to the system PC would ultimately allow an attacker to gain access to p,q but if required considerable physical protection could be given to the system unit.

Generation of seeds

The second condition for the period of the key to be easily determined is that the seed x_0 is chosen such that $\text{ord}_n\, x_0 = \lambda(n)/2$. The seed, however, is chosen by the *sender* of the messages, *not* by the recipient! Without knowledge of $\lambda(n)$ the sender cannot easily check that the above condition holds and he cannot obtain $\lambda(n)$ without factoring n. A severe problem thus arises in that if this criterion is not satisfied, the key period may not be $\lambda(\lambda(n))$ as desired. We have not overcome this limitation. Our attempts at a possible compromise are discussed in the following section.

Encryption

An input message may be either ASCII or binary. When a message is input to the system (either manually or from disk) the encryption module requests the modulus n. Again, this may either be input manually (which is tedious) or fetched from a file of correspondents. In the absence of any information about n other than the assumption that it is a Blum integer, the encryption module requests a prime number of no special form from the prime generator. This is guaranteed to be relatively prime to n and is used as the starting seed x_0. Knowing the bit length of the input message, the encryption module then generates the key sequence. The user is given the option of adding any number of least significant bits from the residues up to the maximum of $\log_2 l$ as discussed on page 3. The module then outputs the XOR of the message with the key stream to a cipher disk file and appends the final residue in hexadecimal form along with an indication of the number of keystream bits added per residue, separated from the bitstream by delimiter sequences.

Currently, our system is used only for encrypting relatively short messages and the module keeps successive residues in memory. Periodically, the module uses *Floyd's cycle finding algorithm* [15] to check that the key sequence has not started to repeat. Floyd's algorithm avoids the need to check every new residue pairwise against every preceding one by the observation that if the sequence $\{x_i\}$ (mod n) is periodic then ultimately this fact will be revealed by the test: Is $x_{2i} \equiv x_i$ (mod n)? If the key starts to repeat, the module issues a warning to the user to this effect. The user is given the option of carrying on and accepting the resulting weakening of security or aborting the process and starting with a new seed.

Clearly, the limitation is on the number of residues (each up to 800 bits long) that can be stored. We have ciphered messages up to about 4 Kbytes (32 Kbits) in length which corresponds roughly to a full page of A4 text. Using 800 bit residues and adding 9 bits to the keystream per residue requires the storage of around 3640 residues. This is only 364 Kbytes of memory which any PC can handle comfortably. However, someone would not be able to use our system to send us the Encyclopedia Britannica in ciphered form!

Decryption

On receipt of an encrypted message, the decryption module scans the file for the delimiter sequences. On finding these, the length of the message, the number of keystream bits per residue and the final residue are read and the module retrieves p,q,a and b from the disk system file. Using these parameters the module evaluates the seed residue x_0 using the algorithm on page 7. The module then uses a similar sequence of operation to the encryption module to recreate the key sequence and output the XOR of the key with the cipher bitstream to disk.

Speed

Currently, the system speed is probably the optimum that can be expected from a pure software implementation. We have not generally used the maximum key sizes possible but worked mainly in the 170 digit range. This corresponds roughly to a factoring time of about 1 million years and is deemed secure enough for most purposes! Using moduli of this size, the generation of a new key takes between 2 and 4 hours depending on whether the algorithm 'gets lucky' or not.

For encryption and decryption, the computation of each residue involves the squaring of a 170 digit number modulo a 170 digit number. Each square mod *n* operation takes around 80mS and so about 12 residues are generated per second. Since 170 digits is approximately 565 bits the possibility exists of appending 9 bits per residue to the keystream. Thus, 108 bits per second of message may be processed. For the 4Kbyte (A4 page) message sizes discussed earlier, the processing time is around 5 minutes.

Speeding the system in hardware

A hardware accelerator for the QRC system is under development. The unit is arranged as a general purpose number theory processor with 1024 bit registers and capable of executing the instructions ADD, SUB, MUL, MOD and DIV - the architecture of the ultimate RISC processor! While using standard 50 nS 32-bit flash multipliers, it will use a new algorithm to evaluate the MOD function using lookup tables in very fast static RAM. The architecture will be heavily pipelined to optimise throughput.

Preliminary analysis of the prototype design indicates that the time required to square a 1024 bit quantity and reduce the result modulo a 1024 bit quantity will be at most 2mS. With 500 residues produced per second and 10 bits appended to the keystream per residue, 5000 bits of message per second may be processed. The processing of a 4K message will then take less than 7 seconds. This represents a speed increase of over 40 times compared with the software implementation in addition to using double the length of modulus. This will allow the cipher to be realistically used for high security online applications as well as making static encryption and decryption very much faster than the existing software.

It is envisaged that the system will be used in the Comprehensive Security System (CSS) [16] developed in the Network Research Group using the *security management centre* concept.

A progress report on this implementation will be published in due course.

REFERENCES

[1] Diffie, W. & Hellman, M.E. *New Directions in Cryptography,* IEEE Trans. Inform. Theory **22** (1976) pp 644-654.

[2] Rivest, R.L., Shamir, A. & Adleman, L. *A Method for Obtaining Digital Signatures and Public Key Cryptosystems,* Comm. ACM, **21**, 1978, pp 120-126.

[3] Rabin, M.O. *Digitalised Signatures & Public Key Functions as Intractable as Factorization,* Technical Report MIT/LCS/TR212, MIT Lab. Comp. Sci. Cambridge, MA, Jan 1979.

[4] Merkle, R.C. & Hellman, M.E. *Hiding Information and Signatures in Trapdoor Knapsacks,* IEEE Trans. Inform. Theory, **24**, 5, Sept 1978, pp 525-530.

[5] Blum, L., Blum, M. & Shub, M. *A Simple Unpredictable Pseudo-random Number Generator,* SIAM Journal on Computing, **15**, 2, 1986, pp 364-383.

[6] Brassard, G. *Modern Cryptology,* Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1988, pp 35-39.

[7] Blum, M. & Goldwasser, S. Proc CRYPTO 89 Santa Barbara, CA.

[8] Knuth, D. *The Art of Computer Programming Vol 2 : Semi-Numerical Algorithms,* Addison-Wesley, Reading, MA, 1981 pp 378-380.

[9] Shanks, D. *Solved and Unsolved Problems in Number Theory,* Chelsea, NY, 1976.

[10] Shamir, A. *On the Generation of Cryptographically Strong Pseudo-random Sequences,* ACM Trans. Computer Systems 1 (1983) pp 38-44.

[11] Yao, D. *Theory and Application of Trapdoor Functions*, Proc. 23rd IEEE Sym. Foundations of Computer Science, 1982, pp80-91.

[12] Vazirani, U.V. & Vazirani, V.V. *Efficient and Secure Pseudo-random Number Generators,* Proc. 25th IEEE Sym. Foundations of Computer Science 1984, pp 458-463.

[13] LeVeque, W. *Fundamentals of Number Theory,* Addison-Wesley, Reading, MA, 1977.

[14] Gordon, J.A. *Strong Primes are Easy to Find,* Proc. EUROCRYPT 84, Springer-Verlag, 1985, pp 216-223.

[15] Riesel, H. *Prime Numbers and Computer Methods for Factorization,* Birkhauser, Boston, 1985, ISBN 0 8176 3291 3.

[16] Shepherd, S.J., Sanders, P. & Patel, A. *A Comprehensive Security System - the Concepts, Agents and Protocols,* Computers & Security, **9**, 1990, pp 631-643.

BIOGRAPHIES

Simon J Shepherd is a lecturer in the Electrical Engineering Department at the University of Bradford. He holds a First Class Honours Degree in Electrical Engineering and is currently completing a Ph.D. in cryptography. His research interests include problems in number theory, integer factoring algorithms and supercomputer architectures to attack problems in computational number theory.

Peter W Sanders is Director of the Network Research Group at the University of Plymouth. Under his direction, the group is currently undertaking several research contracts involving new methodologies in network optimisation, large scale distributed security and secure communications.

Dr Colin T Stockel is a Principal Lecturer in the Computing Department at the University of Plymouth and is closely involved with the work of the Network Research Group. His research interests include computer algorithms and their application to problems in technology.